



AFRL-RI-RS-TR-2018-013

**ASSURED CLOUD COMPUTING UNIVERSITY CENTER OF
EXCELLENCE (ACC-UCOE)**

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

JANUARY 2018

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2018-013 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

LAURENT Y. NJILLA
Work Unit Manager

/ S /

WARREN H DEBANY, JR.
Technical Advisor, Information
Exploitation and Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) JANUARY 2018		2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) MAY 2011 – JUN 2017	
4. TITLE AND SUBTITLE ASSURED CLOUD COMPUTING UNIVERSITY CENTER OF EXCELLENCE (ACC-UCOE)				5a. CONTRACT NUMBER FA8750-11-2-0084	
				5b. GRANT NUMBER N/A	
				5c. PROGRAM ELEMENT NUMBER 61102F	
6. AUTHOR(S) Roy H. Campbell				5d. PROJECT NUMBER CLUD	
				5e. TASK NUMBER UC	
				5f. WORK UNIT NUMBER OE	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Illinois at Urbana-Champaign 201 N. Goodwin Avenue Urbana, IL 61801				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RIGA 525 Brooks Road Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2018-013	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT -Security and isolation in cloud environments -Cyber infrastructure security -Design of algorithms and techniques for real-time assuredness in cloud computing -Map-reduce task assignment with data locality constraint -Trustworthiness estimation for workflow completion -Application-aware cloud network resource allocation					
15. SUBJECT TERMS Security in Cloud environments, cyber infrastructure security					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 119	19a. NAME OF RESPONSIBLE PERSON LAURENT Y. NJILLA
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code)

TABLE OF CONTENTS

SECTION 1	1
SECURITY AND ISOLATION IN CLOUD ENVIRONMENTS.....	1
SECTION 2	6
CONTAINERS: STUDY SECURITY ISOLATION CONCERNS WHEN USING CONTAINER-BASED VIRTUALIZATION AND DESIGN MECHANISMS TO ADDRESS IDENTIFIED SECURITY CONCERNS COORDINATION AND PROBABILISTIC CONSISTENCY.....	6
SECTION 3	44
CYBER INFRASTRUCTURE SECURITY: DYNAMIC POLICY MONITORING WITH INTERFERENCE IN CLOUD...	44
SECTION 4	46
DESIGN OF ALGORITHMS AND TECHNIQUES FOR REAL-TIME ASSUREDNESS IN CLOUD COMPUTING	46
SECTION 5	50
GREATLY INCREASE THE ASSURANCE LEVEL TO CLOUD COMPUTING SYSTEMS THROUGH FORMAL SPECIFICATION & VERIFICATION IN MAUDE	50
SECTION 6	56
DESIGN OF ALGORITHMS AND TECHNIQUES FOR REAL-TIME ASSUREDNESS IN CLOUD COMPUTING	56
SECTION 7	63
MAP-REDUCE TASK ASSIGNMENT WITH DATA LOCALITY CONSTRAINT.....	63
SECTION 8	65
SECURITY AND PRIVACY MECHANISMS: AN ANALYSIS OF CERTIFICATIONS FOR FEDERAL CLOUD SERVICE PROVIDERS	65
SECTION 9	71
SECURITY DATA ANALYSIS AND DESIGN OF SOFTWARE ARCHITECTURE FOR ATTACK CONTAINMENT....	71
SECTION 10	78
TEST-BED FOR EXPERIMENTAL EVALUATION: DESIGN AND PROTOTYPE OF TECHNIQUES FOR PROVIDING CLOUD ERROR AND ATTACK RESILIENCY	78
SECTION 11	91
TRUSTWORTHINESS ESTIMATION FOR WORKFLOW COMPLETION	91
SECTION 12	96
APPLICATION-AWARE CLOUD NETWORK RESOURCE ALLOCATION.....	96
BIBLIOGRAPHY.....	100
LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS.....	111

LIST OF FIGURES

FIGURE.....	PAGE
1: THE APPLICATION SCENARIO AT A HIGH LEVEL	8
2: CONSTRAINTS ENFORCED BY CONVENTIONAL SYNCHRONIZERS.....	21
3: RESOURCE ADMINISTRATION SYNCHRONIZATION CONSTRAINT	22
4: DISABLING ATTACK	23
5: ATOMICITY ATTACK.....	23
6: CONSTRAINTS ENFORCED BY SCOPED SYNCHRONIZERS.....	24
7: INFORMATION LEAK THROUGH	26
8: THE SLIDING WINDOW PROTOCOL IN LANG-A	29
9: SPEEDUP SUMMARY FOR LOCAL & REMOTE EXECUTION OF N-QUEEN	36
10: SPEEDUP SUMMARY FOR LOCAL & REMOTE EXECUTION OF IMAGE PROCESS	37
11: SPEEDUP SUMMARY FOR LOCAL & REMOTE EXECUTION OF N-QUEEN PROB	38
12: SPEEDUP SUMMARY FOR REMOTE EXECUTION VS. LOCAL+REMOTE	39
13: SPEEDUP SUMMARY FOR REMOTE EXECUTION (X REMOTE WORKERS)	40
14: SPEEDUP SUMMARY FOR LOCAL EXECUTION (BASE CASE) VS. REMOTE.....	41
15: OVERHEAD RESULTING FROM ELASTICITY MANAGER FOR IMAGE PROCESSING ..	42
16: ATTACK STAGES AND TRACES OF A MULTI-STAGE ATTACK USING VEMOM	72
17: WORKFLOW OF FACTOR GRAPH FRAMEWORK TO DETECT ATTACKS	74
18: B-BENIGN, S-SCAN, --INITIAL COMPROMISE, G-GATHER INFORMATION, ETC.	77
19: HOOK BASED MONITORING.....	80
20: HYPERTAP PROTOTYPE COUPLED WITH THE KVM HYPERVISOR	83
21: HPROBES INTEGRATED WITH KVM HYPERVISOR.....	85
22: A PROBE HIT IN THE HPROBE PTOTOTYPE.....	85
23: SINGLE PROBE LATENCY.....	87
24: OVERVIEW OF THE APPROXIMATE FAULT LOCALIZATION APPROACH	88
25: EXAMPLE OF AN END-TO-END FLOW	89
26: GENERAL STRUCTURE FOR EVIDENCE-BASED TRYST JUDGEMENT	92
27: HADOOP/YARN SYSTEM PERFORMANCE	93
28: CUMULATIVE DISTRIBUTION FUNCTION OF A MAPREDUCE JOB COMPLETION	94
29: BELIEF NETWORK MODEL	95

LIST OF TABLES

TABLE	PAGE
1: THE SYNTAX OF GLOBAL TYPES	27
2 THE SYNTAX OF LOCAL TYPES.....	30
3 SPECIFICATIONS OF THE USED EQUIPMENT FOR EVALUATION	32
4: PROGRAMS USED TO EVALUTATE OUR FRAMEWORK.....	34
5: CONVERSION FROM RAW LOGS TO EVENTS AND LABELED ATTACK STAGE.....	76
6: CVE-2008-0600 DETECTOR/CRIU	88

Section 1 Security and Isolation in Cloud Environments (Rakesh Bobba, Sibin Mohan, Roy Campbell, and Read Sprabery)

Section 1 Summary of Research Project

Assured and mission critical cloud computing requires security and isolation both at the compute platform and network levels. In this work we focus on providing security and isolation both at the platform level and network level. In particular, at the platform level we focus on a) integrity of software appliances, b) preventing information leakage through cache-based side channels across security domains (e.g., organizations in multi-tenant clouds, security levels in private single-tenant clouds), and c) monitoring and secure logging. At the network level we focus on, a) network update abstractions that is not only consistent but also allows interflow constraints (e.g., spatial and temporal) to be satisfied during the update process, and b) secure and partial delegation of network configuration to tenants.

Section 2 Introduction

Assured and mission critical cloud computing requires security and isolation both at the compute platform and network levels. At the computing platform level, confidentiality, integrity and availability of the data and computations are needed. For instance, one needs to ensure that the software appliances (e.g., single-purpose VMs) are not tampered with or compromised. Similarly, one needs to ensure that information does not leak across domain or security level boundaries. While MLS systems may be able to ensure this at the application and OS levels, shared hardware such as caches still pose a risk by enabling side-channels.

At the network level, temporal and spatial isolation of critical flows across domains or security levels is important in mission critical computing and for compliance. For instance, a cloud infrastructure manager may need to ensure that critical flows belonging to different security levels do not transit the same links/nodes (spatial isolation).

Our work focuses on various security and isolation problems at both the computing platform and network levels. At the platform level, i) we worked on verifying the integrity of software appliances using a whitelisting approach, ii) worked on a framework for preventing information leakage through cache-based side channels across security domains, and iii) worked on secure logging and monitoring frameworks for both security and compliance. At the network level, i) we worked on a novel configuration update abstraction for software define networks (SDNs) that can not only ensure consistency but also take interflow constraints into account during the update; ii) we worked on an initial framework for secure and partial delegation of cloud network infrastructure configuration.

Section 3 Methods, Assumptions, and Procedures

Virtual Appliance Integrity [6,8]: Design a software whitelist-based framework that allows cloud providers and users to determine the trustworthiness of a virtual appliance measured in

terms of its software integrity. The framework assumes that integrity codes for software are available through the vendors and requires virtual appliance image providers/creators to provide a “bill-of-software” for the appliance. This “bill-of-software” is then compared against what is actually found in the appliance image and a trust rating is computed based on the state of the software and the presence of malware. Our empirical study with 151 virtual appliance images from public image stores found that ~9% of them had seriously questionable software integrity.

Cache-based Side Channel Attack Defense [T1]: Proposed a framework to defend Containers/VMs against side-channel attacks by co-tenants in cloud environments. It leverages both hardware and software mechanisms to achieve low overheads. In particular, we leverage Intel’s CAT technology to partition last level cache and provide spatial isolation, co-scheduling to provide temporal isolation, and selective sharing of libraries to balance performance and security needs.

Policy-Based Monitoring for Security [1,T4,5,9]: In our recent work we proposed a framework for event-based logging of virtual applications to enable policy-based monitoring [1]. The framework leverages virtual probes to log system calls. The logs can then be used to detect anomalies using a whitelisted policy. The challenge was in ensuring that logging is complete and not easily circumvented by the adversary.

In the past we proposed a multi-domain policy-conformance monitoring framework for hybrid cloud deployments that limits data exposure [5] when sharing logs for compliance. Specifically, the framework minimizes the logs that need to be shared for policy compliance and monitoring across domain boundaries.

While the aforementioned work focused on log sharing across multiple cloud providers, in [9] our focus was assurance of log information collected by a virtual machine. We undertook a preliminary exploration of an approach where the cloud provider could expose an API to provide information that can help corroborate the information in the logs provided by the virtual machine.

Supporting Inter-flow Constraints during updates in SDNs [2,3,T3]: Given the distributed nature of a network, global configuration changes are not atomic as a result of which the network will have transient configuration states. We proposed a novel SDN update consistency abstraction called “Interflow Consistency” that builds on existing consistent update abstractions and ensures that temporal and spatial flow isolation constraints (e.g., security policy constraints) are respected during such transient states.

Enabling Network Control Delegation in Cloud Networks [4]: We explored the idea of allowing cloud tenants to manage their portion of the network without impacting the security of cloud infrastructure provider so they may manage the network configuration to ensure security policy compliance (e.g., interflow consistency etc.). The approach leverages SDN network virtualization tools such as a FlowVisor for network control delegation.

Section 4 Results and Discussion

Virtual Appliance Integrity [6]: Our empirical study with 151 virtual appliance images from public image stores found that ~9% of them had seriously questionable software integrity. Only about half of them were flagged by traditional malware scanning, demonstrating that a whitelist-based approach is necessary and complementary to traditional blacklisting-based approaches such as signature-based scanning approaches.

Cache-based Side Channel Defense [T1]: Our cache-based side-channel attack defense requires no changes to applications and is suitable for either single-tenant MLS Clouds, multi-tenant clouds or a combination. Our initial prototype showed promising results. A full paper is under preparation.

Event-based Logging and Monitoring for Security [1,T4]: A prototype of our logging framework showed that the overhead is naturally dependent on the number and type of systems events monitored. When monitoring *exec* and *open* system calls the overhead was less than 10% when monitoring Apache. However, when high-assurance for log completeness is needed the overheads went up to 55%. On the other hand, overheads for monitoring OpenSSL were negligible in both cases.

Supporting Inter-flow Constraints during updates in SDNs [2,3,T3]: We implemented a prototype system on a Mininet OpenFlow network and Ryu SDN controller. Experimental results show that our approach is able to enforce inter-flow consistency constraints with reasonable overheads and that overheads for version isolation (temporal isolation) are higher than for spatial isolation. Furthermore, when only spatial isolation constraints are in use, overheads on update times for flows that have no isolation constraints are very small (around 1%).

Section 5 Conclusions

Ensuring security and isolation in cloud computing environments is a challenging problem. Security and isolation need to be addressed at every layer of computing and networking stacks. Thanks to the Assured Cloud Computing (ACC) – University Center of Excellence, we have made significant progress towards better understanding these challenges and towards addressing some of them. However, many challenges remain and a sustained focus and effort are needed to realize the goal of assured and mission critical cloud computing.

Section 6 Bibliography

Conference and Journal Publications

1. Read Sprabery, Zachary Estrada, Zbigniew Kalbarczyk, Ravishankar Iyer, Roy Campbell, and Rakesh Bobba, “Defense in Depth for Virtual Applications Built on Event Based Probing of Untrusted Guests,” *Annual Computer Security Applications Conference (ACSAC 2016)*, Los Angeles, CA, December 5-9, 2016.

2. Weijie Liu, Rakesh B. Bobba, Sibin Mohan, and Roy H. Campbell, "Inter-Flow Consistency: A Novel SDN Update Abstraction for Supporting Inter-Flow Constraints", *IEEE Conference on Communications and Network Security (CNS 2015)*, Florence, Italy, September 28-30, 2015.
3. Weijie Liu, Rakesh B. Bobba, Sibin Mohan, and Roy H. Campbell, "Inter-Flow Consistency: Novel SDN Update Abstraction for Supporting Inter-Flow Constraints", *NDSS Workshop on Security of Emerging Networking Technologies (SENT)* co-located with *Network and Distributed System Security Symposium (NDSS 2015)*, San Diego, CA, February 8, 2015.
4. Salman Malik, Mirko Montanari, Jun Ho Huh, Rakesh B. Bobba, Roy H. Campbell, "Towards SDN Enabled Network Control Delegation in Clouds", *Third International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV 2013)*, co-located with the *43rd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013)*, Budapest, Hungary, June 24-27, 2013.
5. Mirko Montanari, Jun Ho Hun, Rakesh B. Bobba, and Roy H. Campbell, "Limiting Data Exposure in Monitoring Multi-domain Policy Conformance", *6th International Conference on Trust and Trustworthy Computing (TRUST 2013)*, London, UK, June 17-19, 2013.
6. Jun Ho Huh, Mirko Montanari, Derek Dagit, Rakesh Bobba, Dong Wook Kim, Yoonjoo Choi and Roy H Campbell, "An Empirical Study on the Software Integrity of Virtual Appliances: Are You Really Getting What You Paid For?", *8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS 2013)*, Hangzhou, China, May 13-16, 2013.
7. Stephen Skeirik, Rakesh B. Bobba, and Jose Meseguer, "Formal Analysis of Fault-tolerant Group Key Management using ZooKeeper", *First International Workshop on Assured Cloud Computing Conference (CCGrid 2013)*, *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Delft, The Netherlands, May 13-16, 2013.
8. Jun Ho Huh, Mirko Montanari, Derek Dagit, Rakesh Bobba, Dong Wook Kim, Yoonjoo Choi and Roy H Campbell, "Assessing Software Integrity of Virtual Appliances through Software Whitelists: Is it any good?", *Network & Distributed System Security Symposium (NDSS 2013)*, San Diego, CA, February 24-27, 2013.
9. Mirko Montanari, Jun Ho Huh, Derek Dagit Rakesh Bobba and Roy H. Campbell, "Evidence of Log Integrity in Policy-based Security Monitoring", *2nd International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV 2012)*, in conjunction with the *42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, Boston, MA, June 25-28, 2012.
10. Jingwei Huang, and David M. Nicol, Rakesh Bobba, Jun Ho Huh, "A Framework Integrating Attribute-based Policies into Role Based Access Control", *17th ACM Symposium on Access Control Models and Technologies (SACMAT 2012)*, Newark, NJ, June 20-22, 2012.

Theses

1. Mohammad Ahmad, “Cauldron: A Framework To Defend Against Cache-Based Side-Channel Attacks In Clouds”, MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2016.
2. John Bellessa “Implementing MPLS with Label Switching in Software Defined Networks”, MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
3. Weijie Liu, “Inter-Flow Consistency: Novel SDN Update Abstraction For Supporting Inter-Flow Constraints”, MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
4. Read Sprabery, “An architecture for trustworthy services built on event based probing of untrusted guests”, MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, August 2016.

Section 2 Containers: Study Security Isolation Concerns When Using Container-based Virtualization and Design Mechanisms to Address Identified Security Concerns, Coordination, and Probabilistic Consistency (Gul Agha, Reza Shiftehfar, Minas Charalambides, and Kirill Mechitov)

Section 1 Summary of Research Project

Smart sensors, mobile devices, and cloud-based ecosystems are increasingly integrated to create a mobile cloud. However, integration can lead to security and trust issues. For example, security of cloud spaces has sometimes been breached by accessing peripheral devices such as a HVAC system. Our research shows how cloud security and trust can be improved in mobile cloud systems—thus facilitating Assured Cloud Computing (ACC). Specifically, we have developed a programming framework that can simplify development of mobile cloud systems while providing assurance in satisfying service level agreements and constraints of mobile devices such as bandwidth, processing power and energy. The framework builds on research in actor programming languages, constraint systems, and runtimes to support mobility. Experiments with a prototype implementation of the framework illustrate its utility. Finally, we show how the framework may be extended to facilitate formal methods for reasoning about ACC systems using recent work in multi-party session types and applications of learning to infer concurrency patterns and coordination constraints.

Section 2 Introduction

Mobile devices and smart sensors have become ubiquitous. Yet, to realize their full potential, they need to be integrated into a broader context by interacting with network services—in particular, those offered by computing clouds, which can provide elastic on-demand access to virtually unlimited resources at an affordable price. We call a system that achieves this integration a *mobile cloud*.

An important issue that affects mobile cloud users, and which currently precludes availability of many complex applications, is the multitude of limitations on resources of mobile devices. Compared to laptops and desktops, mobile devices typically have weaker hardware, more restricted network access, and more limited availability of energy. However, even in the face of such constraints, users require availability and timeliness of services, device efficiency, and assurance of security and privacy.

We show how these requirements can be met in mobile clouds by proper use and coordination of cloud resources. More specifically, we have developed a programming middleware framework, called *IMCM*, which with minimal developer effort can dynamically outsource storage and computation needs of demanding mobile applications to cloud spaces. To achieve such off sourcing, certain parts of mobile applications are selected, sent to the cloud space, executed, and

the results brought back to the mobile device. This process is known as *code offloading* and has been widely studied within the context of distributed systems and grid computing.

Code offloading can be either *coarse grained*, e.g., at the level of virtual machines, or *fine grained*, at the level of components or individual computations. Our framework supports fine-grained offloading, which has greater potential for improvements in energy use and response times of mobile devices. The framework considers a mobile-cloud application as a composition of self-contained autonomous actor components, which are individually subject to performance and energy consumption monitoring. For energy consumption specifically, technique we consider relies on statistical comparison of energy drops for actors of different types against a control setting on the mobile device. This is in contrast to existing frameworks, which typically rely on device-level energy measurements to make offloading decisions.

While code offloading can improve application user experience and device resource usage, it must be performed while respecting security and privacy requirements. In an environment with both trusted (private) and untrusted (public) cloud resources, the origin and destination of data and code sent from devices, e.g., during code offloading, must be taken into account. To support such *hybrid cloud* environments, the framework allows specifying detailed *security policies* that are monitored and enforced by the application runtime in the cloud. In addition, runtime monitoring is used to collect observations on application intent, which in turn can be used to infer and adapt (constrain) behavior of application components, e.g., when past communication patterns suggest offloading to specific cloud resources for better application latency. Since applications at runtime are collections of actors, application constraints, explicitly expressed or inferred, can be encoded in the framework in a variety of formal representations, such as actor synchronization constraints and actor session types, to allow refinement and synthesis of new constraints.

By using these approaches and techniques, our framework facilitates holistic Assured Cloud Computing (ACC) for hybrid mobile clouds.

Consider a mobile application that should perform *facial recognition* of a given image using a database of known faces, of which some must remain confidential. Since this kind of image processing is computationally expensive, tasks should be offloaded to the cloud whenever possible. We assume application developers want to deploy this application in a *hybrid* cloud environment, spanning both a public and a private cloud. Using existing frameworks, engineers face a number difficult issues in the development, deployment, and maintenance of such an application:

Productivity. The application may have to be decomposed in specific ways to enable fine-grained code offloading, and the decomposition may be different depending on the

typical deployment scenarios. Developers may have to translate high-level application requirements into executable imperative code. To programmatically access sensor data, knowledge of low-level interfaces may be required.

Security and Privacy. To achieve requirements on security and privacy, developers may have to use specific knowledge about the deployment environment, e.g., whether a specific offloading task is sent to a certain public cloud. Developers may also need to add security checks at specific places in the application code, e.g., where a photo that should remain confidential is accessed.

Maintainability. The application may have to be re-architected and re-deployed due to small changes in the environment, e.g., cloud provider changes or increases in average network latency. When application requirements on energy consumption and availability change, developers may have to manually adjust parameters inside imperative code.

A central goal of the IMCM middleware framework is to mitigate these and related issues. Specifically, by programming the application using the actor model, there is no particular tie to a specific code offloading approach, although actor granularity matters for offloading efficiency. When requirements are encoded as declarative constraints enforced by the framework, application evolution becomes less involved and prone to failures; developers no longer carry the burden of inserting code for checking security policy conformance. The framework also hides low-level sensor interfaces. In addition, programmers need not write any logic for deciding when it is beneficial (with respect to energy consumption, latency, etc.) to offload actors into the cloud. Instead, using data on energy consumption, latency times, policies, and other runtime information, the framework can make offloading decisions on-the-fly.

Figure 1 illustrates the application scenario at a high level when the IMCM framework is used. The image application runs on one or more mobile devices that may offload certain actors to either the private or public cloud. Meanwhile, the framework runtime performs monitoring of devices and can provide the data to determine when it is appropriate perform offloading.

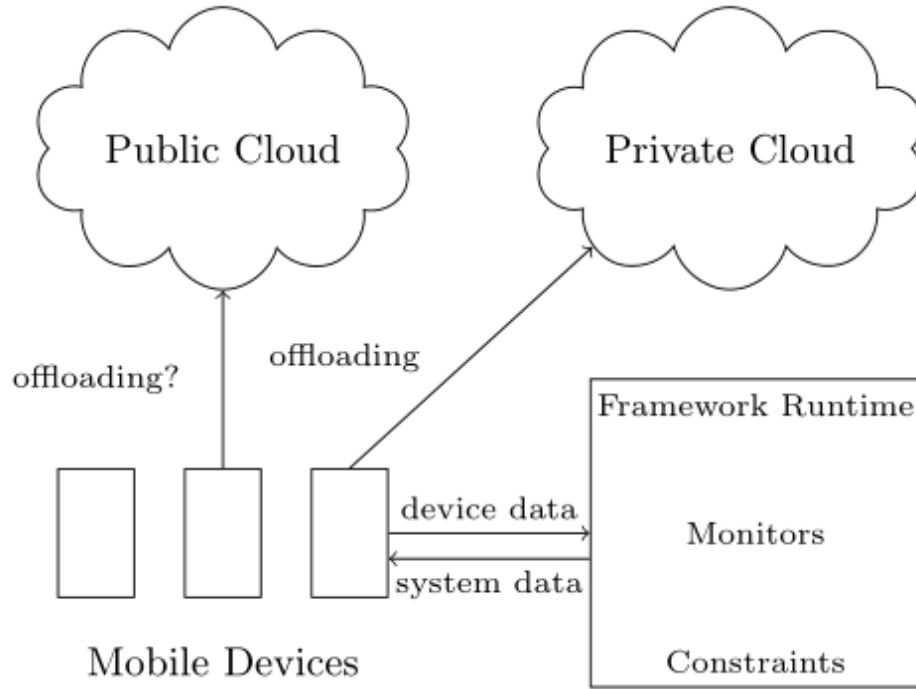


Figure 1. The application scenario at a high level when the IMCM framework is used

Section 3 Methods, Assumptions, and Procedures

Our framework is focused on improving individual application performance while addressing dynamic run-time environment, end-user context, and application behavior. Unlike previous research, our system supports offloading to multiple remote locations, a concurrent application model, and simultaneous execution on both mobile device and remote cloud resources.

IMCM Framework Overview

Many organizations, developers or users benefitting from cloud resources have privacy requirements, expectations, and policies in terms of how different private or public cloud resources can be used by a mobile application. Without having enough flexibility in the offloading framework to address these requirements, many users will not be able to benefit from the cloud resources. In order to accommodate these requirements, we describe a language to define policies, and explain how the framework can be customized to address them.

While addressing these policies is critical, other quantitative properties such as performance and energy characteristics on the mobile device greatly affect the quality of an application in meeting overall user requirements. The framework allows configuring policies that need to be enforced, but they may affect the performance and energy usage of the application. On the other hand, optimized performance and energy can possibly be leveraged for providing stronger privacy

guarantees. Through the IMCM framework, we discuss mechanisms that allow developers or users to control all privacy, performance and energy aspects of their applications via code offloading. In order to formulate the application component offloading problem, a comprehensive mobile-hybrid-cloud application model is needed.

Cloud Model

Over time, cloud services have moved from the model of using public cloud spaces to private clouds and recently to the hybrid model combining both. Cloud infrastructure is traditionally provided by large organizations, thus referred to as public clouds. However, storing data on third-party machines suffers from potential lack of control and transparency in addition to the legal implications. Cryptographic methods can be used to secure the data by encrypting it before storing it in a public cloud, while decryption keys are only disclosed to authorized users. However, these solutions do not scale well and inevitably introduce heavy computational overhead.

In modern mobile-cloud applications, application code is also stored along with data in the cloud. This creates an additional challenge with using public clouds, wherein encrypted pieces of code cannot be executed without decrypting and revealing its content to the cloud provider. These issues have caused companies to gradually move toward building their own private clouds. However, owning private datacenters is not as efficient, reliable, or scalable as using the public ones. Thus, in recent years, a combination of both private and public cloud spaces is used that benefits from all the advantages of the public cloud while keeping the confidential or sensitive data and algorithms in-house. Unlike previous mobile-cloud solutions that consider only one single remote location for offloading, our model considers a hybrid cloud space consisting of one or several private and public cloud spaces and allows concurrent application component offloading and execution on all of them.

Cloud Application Model

In order to replace the traditional data-centric view of the cloud with a more general data/computation-centric view, the current popular *service-oriented architecture* that provides services on data stored in the cloud to external users, needs to be replaced with a new architecture that dynamically and transparently leverage cloud resources to address end-user mobile device limitations. An *elastic application development environment* allows components storing data or performing computations to be transparently distributed between private clouds, public clouds, and end-user device. When such an application is launched, an *elasticity manager* monitors the environment, measures resource requirements of different application components, and makes decisions about component distribution between mobile device and different cloud spaces based on run-time parameters, application behavior, and user expectations. This allows

mobile applications to enforce security policies while adapting to different workloads, performance goals, energy limitations, and network latencies.

In order to prevent creation of additional work for application developers, unnecessary details of distribution and move-around of application components should be masked. In order to reach the maximum level of parallelism without the hassle of traditional multi-threading model, modern cloud-based applications avoid using a shared memory model that is unnatural for developers and leads to error-prone non-scalable programs. Instead, modern cloud-based applications restrict the interaction between various components to communication using messages. This approach to cloud application development aligns with the concepts of the *actor model of computation* that sees distributed components, called actors, as autonomous objects operating concurrently and asynchronously. In response to a received message, an actor can make local decisions, create new actors, send more messages, or change its behavior to respond differently to the next received message. Compared to the traditional shared memory model, actors are a better fit for highly dynamic applications operating in open and challenging environments. Actors may be created and destroyed dynamically, they can change their behaviors, and migrate to different physical locations. The model provides *natural concurrency*, *resiliency*, *elasticity*, *decentralization*, *extensibility*, *location transparency*, and *transparent migration* that ease the process of scaling-up or -out, which is a critical requirement for cloud-based applications.

Also note that minimizing energy usage on the mobile device requires solving the problem of attributing energy consumption to components of an application. The actor model also lends itself naturally to defining the granularity for energy monitoring at the level of individual or groups of actors. Actor instances can be the primitive units for targeting energy measurements, while groupings of actors of a particular type can be considered for aggregations/higher-level metrics like average energy consumption. Schedulers for actor-based languages also view actors as basic computational entities for scheduling decisions, so the underlying runtime can be instrumented to track actors running in different time intervals.

As a result, our view of a mobile-cloud application consists of actors distributed between local mobile device and different cloud spaces.

Defining Privacy for Mobile Hybrid Cloud Applications

We want to enable developers/users to restrict access to different resources and mobility of sensitive or confidential components resources based on required policies. This requires the framework to follow authorization rules defined by the organizations, developers, or users. Elastic application components on the cloud should adhere to the property of least privileges. Which permissions a component should have may depend on its execution location, application

requirements, or user concerns. Implicit access to device resources may require additional scrutiny when the component is no longer running local to the device. A comprehensive security solution requires authentication, access control, and auditing. There has been significant amounts of work on authentication and auditing for cloud applications in the past, and the existing solutions are mature enough to address most applications. So we focus on an approach for adding policy-based privacy to our framework which restricts the accesses, actions, and mobility of components.

Design of the Authorization System

Since we want to provide fine-grained authorization systems for application components, we adopt a hierarchical approach where organizations can enforce an organization-wide policy while developers and end-users can fine-tune it. An organization is the primary owner of the data and resources and must be able to keep private and public cloud components separate from each other and define an overall policy in terms of resource usage for different users or different applications. Specific applications may also need to further tighten these organization-wide policy rules. End users or programmers must also be able to further restrict resource usage and component distributions for specific applications. As a result, our framework supports two types of policies: *hard* policies and *soft* policies.

A hard policy refers to organization-wide authorization rules defined per user or application by the organization. Users include different developers inside the organization in addition to external clients. On the other hand, a soft policy refers to application-specific authorization rules defined in addition to the organization-wide hard policy. Despite the fact that these two types of policies have complementary roles in increasing system flexibility, a soft policy can only tighten the organization-wide policy and not vice versa. In other words, if the organization-wide hard policy allows a specific user or a specific application to access resources A and B, soft policy can only further restrict the access to one of the resources A or B and can never loosen the restrictions by allowing access to a new resource such as C. Separating the restriction policy definition from the application logic in this way allows organizations to define its hard policies without programmers having to worry about compromising the pre-defined organization-wide policy.

Each application instance initially authenticates itself with a *Policy Manager Machine* (PMM) and receives a locked unchangeable hard policy that contains the organization-wide authorization rules defined by the organization. Each organization can define its authorization policy as one policy for all users, one policy for all applications, one policy per application, one policy per user, or one policy per application instance. In the end, each application instance can acquire one locked hard policy from the policy manager machine. In addition, each application instance can have one soft-policy. Developers can define the initial soft-policy per application or per

application instance. They can also allow end-users to change all or part of this soft policy through the application. To implement these rules, we follow the XACML usage model and assume a *Policy Enforcement Point* (PEP) as part of our elasticity manager. PEP is responsible for protecting authorization rules, sending a request containing description of the attempted action to a *Policy Decision Point* (PDP) for evaluation against available hard and soft policies. The PDP evaluates the requested action and returns an authorization decision for the PEP to enforce.

Our authorization framework needs to be able to apply the restriction rules at the granularity of actors. It still allows defining those rules at higher-level entities, such as groups or sets of actors, but it recursively propagates all those specified authorizations (permissions or denials) to all actors contained within that set at runtime. This makes it easy to specify authorizations holding for a larger set of actors (on the whole system in case ALL is used) and have it propagated to all the actors within that set until stopped by an explicit conflicting restriction rules. Actor frameworks allow multiple actors to be placed together in a container, called actor system or theater, to share common attributes. We respect this structuring in our language and allow authorization rules to be defined on actors, actor systems, sets of actors (called *Group*), set of actor systems (called *Location*), or subset of multiple actors and actor systems (called *Selection*).

While access control models restrict access to different components or resources, our mobile hybrid cloud framework provides more than access restriction. The actor programming paradigm allows an actor to send and receive messages, create new actors, or migrate to new locations. As a result, our authorization grammar must allow defining rules regulating all these actions. Note that these actions are usually bidirectional, meaning that if actor 1 is allowed to send to actor 2, then actor 2 must also be allowed to receive from actor 1 in order for the policy to be consistent. If any of these two actions are not explicitly allowed as part of the policy, the framework automatically rejects both actions, as they will always happen together.

Mobile Hybrid Cloud Authorization Language

Authorization decisions are made based on the attributes of the requester, the resource, and the requested action using policy-defined rules. As a result, defining an authorization policy means defining the authorization entities and their required attributes in addition to defining rules and desired rule orderings.

In the cloud application model where actors are the smallest entities in an application, actors are the finest granularity on which we can define access restriction. In order to provide location transparency, multiple actors running on one runtime instance on one machine are placed inside a container, called actor system or theater as in the SALSA language. Our language supports defining both actors and actor systems. Every actor is defined by its related reference, logical

path to reach the element in the runtime environment, in addition to its containing actor system. The authorization framework uses these attributes to bind the actors defined in the policy to their real-world application components.

In our language, every actor belongs to an actor system. An actor system is defined by specifying its related URL/IP address and the listening port number. Since more than one actor system can run in one runtime instance on a specific machine, both an URL and a port number is needed to connect to different actor systems running on the same machine. Note that the use of actor systems hides all the underlying details such as using thread pools for the use of actors, scheduling the actors, etc., from the programmer or the authorization policy writer.

In order for our language to be able to account for the existence and activities of to-be-developed application-specific components (while enabling writing organization-wide policies), anonymous types of entities are defined as part of the proposed language grammar. A rule called *anonymous-actor* allows restricting the creation and number of unknown actors in a reference-actor-system. Similarly, a rule called *anonymous-actor-system* allows controlling the creation and the number of unknown actor-systems.

Grouping, Selection, and Binding

Although definitions like those in the previous section can be used to define individual actors and actor systems, in many cases it is easier to group several entities and treat them as one. A *Group* definition puts several actors together into one virtual container and allows placing both known actors and unknown anonymous-actors together into one group. Similarly we can have a *Location* definition to provide the same grouping functionality but for actor-systems. One or several previously defined actor-systems, locations or even unknown anonymous-actor-systems can be placed into one container location entity.

Instead of specifying individual entities to form a container, a *Selection* definition can be used to pick entities based on a condition. In order to bind previous dynamic actors and actor-systems to specific run-time component, an *Assignment* definition can be used. Any remaining unbound dynamic actor or actor-system is in passive state and will be ignored while enforcing the policy. Assignment definition can then be used to bind them to specific actors or actor-systems and change their passive state to active at any time.

Policy Description

The main goal of writing a policy file is to define required authorization rules on actions among actors. Previous defined grammar allows defining entities and grouping or selecting them that is

a pre-requisite for defining restriction rules. We now look at using them to express authorization rules and their evaluation ordering.

Each rule definition regulates one action from subject entities to be performed on object entities. Actions include all allowable actions within an actor framework: sending, receiving, migrating, and creating. This allows regulating actions, move-around, and communication between actor components of a mobile hybrid cloud application.

Policy Evaluation

In a mobile hybrid cloud framework with authorization restrictions, every requested action by the subject has to be approved by the authorization framework before being performed on the object. To make a decision, authorization system has to evaluate the defined policy rules. However, it is possible for different policy rules to contradict each other, as rules are human-defined by different parties, organization and developers, at different times, at different levels, and for different purposes. Our framework prioritizes hard policy rules, defined at a higher level by the organization, over soft policy rules, defined by programmers for individual applications or instances. Prioritizing hard policy restriction rules over soft policy rules allows resolving any potential conflict between hard and soft policies. In other types of conflicts between rules of the same type, we always prioritize action denials over permissions.

Every authorization rule can be summarized as a five-tuple of the form $\langle Subject, Object, Action, Sign, Type \rangle$. Here, *Subject* and *Object* are the entities between which the specific action is being restricted. *Sign* can be allowance (+) or prohibition (-) and *Type* covers hard policy (H) or soft policy (S). In order to decide on any requested action, the authorization system has to process rules in a meaningful way from the most prioritized one, usually the most specific rule, to the least prioritized one, the most general one.

Performance and Energy Usage Based Code-Offloading

Target offloading goals can affect the component distribution plan in a hybrid cloud environment with multiple public and private cloud spaces in addition to fully parallel application execution. Thus we examine application performance and energy usage on mobile device as target offloading goals and create an offloading decision-making model for the same.

Migrating an entire VM to a more resourceful machine is expensive, so we consider an offloading process that consists of decision making about appropriate parts of an application to offload in addition to migrating them, executing them on remote servers and bringing back the results. The actor-based mobile-cloud application model provides natural application partitioning and masks component migration process. What remains is finding appropriate components for

offloading and this section focuses on making such optimal offloading decision with respect to target goal, application behavior, and run-time parameters.

Offloading for Sequential Execution on a Single Server

Considering the offloading process cost and its effect on application behavior, we see that such a cost highly depends on the target offloading goal. Offloading goals can vary significantly based on the application or user and range from maximizing the application performance (e.g. games, vision-based applications) to minimizing energy consumption on the mobile device (e.g. background applications).

Equations 1 and 2 below show the offloading goals for maximizing application performance and minimizing energy usage on mobile device respectively. First, let

T_{device} = time for offloadable work to be done on mobile device

T_{transfer} = time for data to be transferred from mobile device

T_{remote} = time for offloadable work to be done on remote server processes

then:

$$T_{\text{device}} > T_{\text{transfer}} + T_{\text{remote}} \quad (1)$$

Second, let

E_{active} = energy spent for carrying out computation on mobile device

E_{transfer} = energy spent by device to transfer data to remote server

E_{idle} = energy spent in idle mode waiting for offloaded work to complete

then:

$$E_{\text{active}} > E_{\text{transfer}} + E_{\text{idle}} \quad (2)$$

The above equations lead to the pause-offload-resume model, which results in sequential execution. We consider parallelism where multiple remote servers are working concurrently with mobile devices. Also note that Equations 1 and 2 are very similar and usually result in close decisions, if power consumption on mobile device for computation, transferring data to remote server and waiting in idle mode are all proportional. This is the case for sequential execution and is the result of assuming mobile screen is to be on even in idle state.

Offloading for Parallel Execution on Hybrid Clouds

Deciding on an optimized offloading plan for parallel applications in a hybrid cloud environment requires considering the application type, available resources at different remote machines, and offloading effects on future application behavior.

Fully parallel execution refers to both parallel execution on multiple remote locations and simultaneous local and remote execution. As a result, the total application execution time is the maximum time required for any of the mobile or remote spaces to finish executing program code

for all of its assigned components. Since local communication between components located on the same machine is relatively fast, we can ignore local communication and only consider communications between components placed at different locations. The offloading goal can be summarized as maximizing application performance ($MaxAppPerf$) or minimizing application execution time ($MinAppExec$) using:

$$\begin{aligned} \max(MaxAppPerf) &= \min(MinAppExec) = \\ \min(\max_{0 \leq L \leq M} (ExecAtLoc(L)) + \max_{0 \leq L \leq M} (CommAtLoc(L))) \end{aligned} \quad (3)$$

A mobile application consists of N components, and each component $i \in [1, N]$ is located at $Loc(i, t)$ at time t . Having M different cloud spaces results in $Loc(i, t) \in [0, M]$ where 0 represents the local mobile device and $[1, M]$ corresponds to different cloud spaces. Assuming that we know the application component distribution between the local mobile device and the hybrid cloud spaces at time t_1 , our goal is to find the component distribution for the next time interval t_2 such that application performance is maximized.

Thus, different parts of Equation 3 can be extended so that the first term $\max(ExecAtLoc(L))$ captures the maximum across M different cloud spaces, of execution time for all components on each of those locations L . This can be obtained using monitoring and previous profiling for execution time of each component in its location at time t_2 .

Similarly the second term $\max(CommAtLoc(L))$ of Equation 3 captures the maximum required time for one of the locations to send out all its communications. This can be obtained using the profiled amount of communication between each pair of components during elasticity manager's running time interval Δ and the location of components across locations in time t_2 .

However, not all components of an application are offloadable. So, a few constraints must be added to the above optimization problem. As we are considering a hybrid cloud consisting of multiple private and public cloud spaces, application developers or users can specify additional constraints in terms of how different components can be offloaded to different locations. These additional constraints can also address privacy issues in terms of not offloading sensitive or confidential components to public cloud spaces.

Let us now examine the differences in terms of minimizing mobile device energy consumption, instead of performance. This goal can be defined as below. Let

E_{app} = application mobile energy consumption

E_{device} = energy saved on mobile device

E_{remote} = total mobile energy saving by remote component execution

E_{rcomm} = energy loss due to local communication becoming remote communication

E_{lcomm} = energy saved due to remote communication becoming local communication
then:

$$\min(E_{\text{app}}) = \max(E_{\text{device}}) = \max(E_{\text{remote}} - E_{\text{rcomm}} + E_{\text{lcomm}}) \quad (4)$$

E_{remote} in Equation 4 can be further elaborated into

$$\sum_{i=1}^N (\text{LocEQ}(0, \text{Loc}(i, t_1)) * (1 - \text{LocEQ}(0, \text{Loc}(i, t_2))) * \text{Energy}(i)) \quad (5)$$

where $\text{Energy}(i)$ is the profiled energy consumption of component i running locally on the mobile device during the time interval Δ , $\text{LocEQ}(l_1, l_2)$ returns 1 if two given locations are identical and 0 otherwise. Note that the first term of the equation considers only components that are currently on the device and second term adds the condition that those element must now be at a remote location. This way energy saving is only counted for components that have been migrated from the local device to a remote location. It should be noted again that our goal is to minimize energy consumption at the mobile device and not the total energy. Thus, the migration of components between remote locations does not help with this goal and is not considered in the equation.

E_{rcomm} and E_{lcomm} in Equation 4 be obtained using the profiled amount of communication between each pair of components and the profiled mobile power when communicating with remote servers.

Similar to its performance counterpart, we can add constraints such as offloading components to remote locations to save local energy should not affect the performance of the application. In other words it allows energy saving as long as a certain service performance quality is satisfied. An important observation we made in our fully parallel application model is that the results of our offloading goals are very different for application performance improvement and energy savings on mobile device. This is unlike the sequential case in which the models lead to similar configuration results. So we use the constraints to add restrictions on how much improvement for one goal can affect the other.

Energy Monitoring

A big challenge to solving Equation 5 is the use of $\text{Energy}(i)$. As mentioned, $\text{Energy}(i)$ is the profiled energy consumption of component i running locally on the mobile device. This requires fine-grained profiling of energy consumption per application component on mobile device. However, most mobile devices do not provide any tool for direct measurement of the consumed energy. Almost all previous research in this area rely on external power meters to measure energy consumption. Although using expensive external power meters work for experimental settings, we cannot expect end users to carry such a device with themselves to profile energy

consumption of the mobile device. This is a big challenge for optimizing energy consumption of mobile hybrid cloud applications. Even if the total energy consumption of the mobile device can be measured, there are multiple applications running on a mobile device at any time. This requires distribution of the total measured energy among those applications. Further there are multiple components within our target application running over times and distributing energy further among those application components is a challenge. The solution we explore here can scalably profile runtime energy consumption of the application, while treating it as a black-box. This approach can detect complex component interactions/dependencies between components or actors in an application that affect energy consumption on mobile device.

We consider mobile applications written using the actor model based programming language SALSA that natively supports migration of actors between mobile and cloud platforms. We build the mechanism to profile running applications from underlying SALSA runtime layer to attribute battery drops to subsets of actor types. It begins with instrumentation of SALSA runtime that enables determining actors scheduled in the application at each (pre-defined) interval of time. Using the corresponding battery drops in these intervals, a combination of linear regression and hypothesis testing techniques is used to infer battery drop distribution of subsets of actor groups within an execution context.

Note that different subsets of actors would be active in each interval so if we observed this data for an application coming from large number of smartphones, it would then be possible to collect measurements that help generate a distribution for battery drop characteristics for different actor types with increasing accuracy. Apart from speeding up the availability of battery drops for subsets of actor types, this crowdsourcing based approach could handle noise in the sensor readings. We would have to partition this data by execution context however, which includes hardware context such as screen or GPS being turned on/off, along with software context such as other running applications on the device. The additional data allows us to manage heterogeneity of context in which different applications are running before being able to do energy attribution. We leave this crowdsourcing based monitoring approach as an extension for future work.

Security Policies and Energy Monitoring

We can now extend IMCM to include energy policy based authorization system which can enforce runtime restrictions on actors running in mobile-cloud ecosystem such as:

- Policies that prevent malicious actors from draining battery on mobile devices to prevent secure actors from carrying out their tasks.
- Energy consumption based policies to restrict sending or receiving of messages from abusive actors and managing DoS attacks by enforcing maximum energy threshold based restrictions on actor creation within a container.

- Organization-wide policies for abusive actors (based on energy characteristics) as the actor signature. This is useful when runtime actor information is unavailable while writing such policies.
- Track the energy consumption of an actor over time, in order to detect any large deviations in energy characteristics that may occur due to the actor being compromised.

Actor-Based Coordination with Synchronizers

Synchronizers are collections of declarative synchronization constraints that can be imposed on groups of actors. The constraints express under which conditions an actor is able to handle a message. Until the conditions are met, the message stays in the actor's mailbox. The constraints have a global effect and affect all messages an actor receives.

The conventional form of synchronizers supports disabling and atomicity constraints. Disabling constraints prevent an actor from handling messages that match a given pattern. For example, by disabling the handlers for all but the initialization message, a disabling constraint ensures that an Actor dispatches (starts to process) the initialization message before it dispatches any other message. Atomicity constraints coordinate groups of actors by bundling messages into indivisible sets. A constraint enforces that either all the messages in a set are dispatched, or none of them are (there is no partial delivery). The constraint provides spatial atomicity.

Programmers declare synchronizers as templates. Similar to classes or actor behaviors, these templates are dynamically instantiated at runtime with concrete values filled in for the parameters. Consequently, synchronizers can adapt the system to meet new specifications during system execution. Actors can install synchronizers at any of their acquaintances. Synchronizers can have local state that changes with the observed messages. They may also overlap, that is, multiple synchronizers can constrain the same actor. Figure 2 shows the effects of a possible synchronizer.

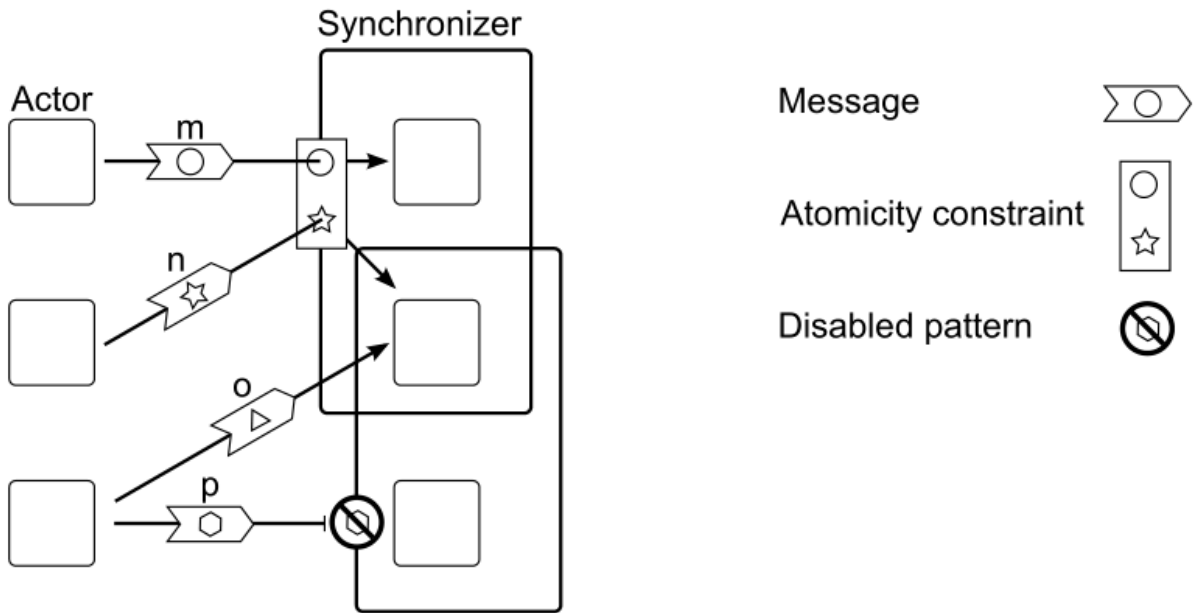


Figure 2 Constraints Enforced by Conventional Synchronizers

Synchronizers support (combinations of) atomicity and disabling constraints. Atomicity constraints ensure that a set of messages is dispatched as a whole and without temporal (happened before) ordering. Messages *m* and *n* satisfy the atomicity constraint together and are therefore dispatched at their target actors. Message *p* matches a disabling pattern in the lower synchronizer and therefore cannot be dispatched.

Consider a system that provides two kinds of resources for its users, for example disk drives and optical drives. There are multiple instances of both drive types and each of these resource kinds is governed by an administrating Actor that limits the number of instances that can be used at the same time. Suppose that the disks and optical drives are accessed over the same network connection. To ensure that drive accesses stay within the bandwidth limit, the administrating actors have to restrict the *total* allocations made of both drive types. The synchronizer in Figure 3 implements the necessary coordination pattern using disabling constraints. It stores the total number of allocated drives in the system in an internal counter *alloc*. Observing requests and releases at the resource administrators updates the counter (lines 5 and 6). When the maximum number of drives has been requested, the synchronizer disables the request handlers of both administrators (line 4). Thus, neither administrator can process further allocation requests. These pending requests can be processed only after one of them releases a drive.

```

1 AllocationPolicy(adm1, adm2, max) {
2   init alloc := 0
3
4   alloc >= max disables (adm1.request or adm2.request)
5   (adm1.request or adm2.request) updates alloc := alloc + 1,
6   (adm1.release or adm2.release) updates alloc := alloc - 1
7 }

```

Figure 3. Resource administration synchronization constraint

Synchronization Constraints in Large-Scale Systems

Scalable coordination models must not only use additional resources efficiently, but also address the inherent requirements of large systems:

Support of dynamic reconfiguration and adaptation. Large systems, for instance a cloud computing service, are expensive to reboot. Nevertheless, the environment and specifications of the system are likely to change over the system lifetime, for example when new services are introduced. A scalable coordination model must therefore support dynamic adaptation.

Robustness against misbehaving actors. The chance of having a faulty, compromised, or malicious actor in a system increases with the system size. A scalable coordination model must therefore be able to cope with uncooperative actors and gracefully degrade in the presence of failures. It must also guard its reconfiguration mechanisms against abuse.

The second requirement implies that, in general, actors in large systems cannot rely on the good intentions of other actors. We therefore think of actors as being mutually suspicious, that is, they do not trust each other. Consequently, actors must try to give others as little control over themselves as possible and follow the principle of least authority. In particular, actors must try to avoid making their (eventual) computational progress dependent on others.

Mutual suspicion conflicts with the global scope of synchronization constraints defined in the conventional synchronizer semantics. Under these semantics, synchronizers observe and affect all messages a constrained actor receives. Since any actor may install synchronizers on acquaintances, malicious actors can cause intentional deadlocks on other actors, effectively resulting in a denial of service at the target. For example, suppose that an actor *A* can handle messages of type *message1*, *message2*, and so on, up to *messageN*. A malicious actor *M* can prevent *A* from receiving any further messages by installing a synchronizer, shown in Figure 4 that disables all message handlers in *A*.

```

1  DisablingAttack(a) {
2    true disables (a.message1 or a.message2 or ... or a.messageN)
3  }

```

Figure 4. Disabling attack

Similar problems arise from atomicity constraints. If M forces A to only dispatch messages in unison with an anonymous actor that never receives any messages, then A will deny all service. An example of such a synchronizer is shown in Figure 5.

```

1  AtomicityAttack(a, anonymous) {
2    atomic( (a.message1 or a.message2 or ... or a.messageN),
3           anonymous.message )
4  }

```

Figure 5. Atomicity attack

Scoped Synchronization Constraints

The examples in Figure 4 and Figure 5 demonstrate that allowing synchronizers to constrain all messages an actor receives is problematic in large systems. We now describe a scoping mechanism for synchronization constraints that restricts their effects to a subset of messages, which addresses this problem.

The central idea behind the approach is that synchronization constraints restrict not the receivers, but the sources of messages. Consequently, a constraint installed on actor A by actor I should not apply to all messages that A receives. Instead, the constraints should only apply to messages received by A if they were sent by actors that are under control of I . Hence, the constraints should only apply if the installing actor I has the capability to impose constraints on the sending actors.

Synchronization constraints, and thus synchronizers, work in the opposite direction of object-capabilities. Object-capability security is the natural security model of actor systems. Its defining notion is that once an actor address—the capability for this actor—is known, any message may be sent to it. Access to services hence depends on the knowledge of actor addresses; security can be implemented through their careful distribution. The underlying assumptions are that addresses are unique across the system and cannot be guessed. For actors, the only ways of obtaining knowledge of other actors' addresses are (1) initialization: the system starts with this knowledge distribution; (2) parenthood: creating a new actor yields an address; and (3) introduction: addresses are values and can be propagated inside messages.

As a complement to object-capabilities, we introduce synchronization-capabilities that determine the scope of synchronization constraints. Synchronizers can constrain messages only if they hold the synchronization-capability to the message source. They receive their synchronization-capabilities from the installing actor. Figure 6 shows the scoping effects of synchronization-capabilities.

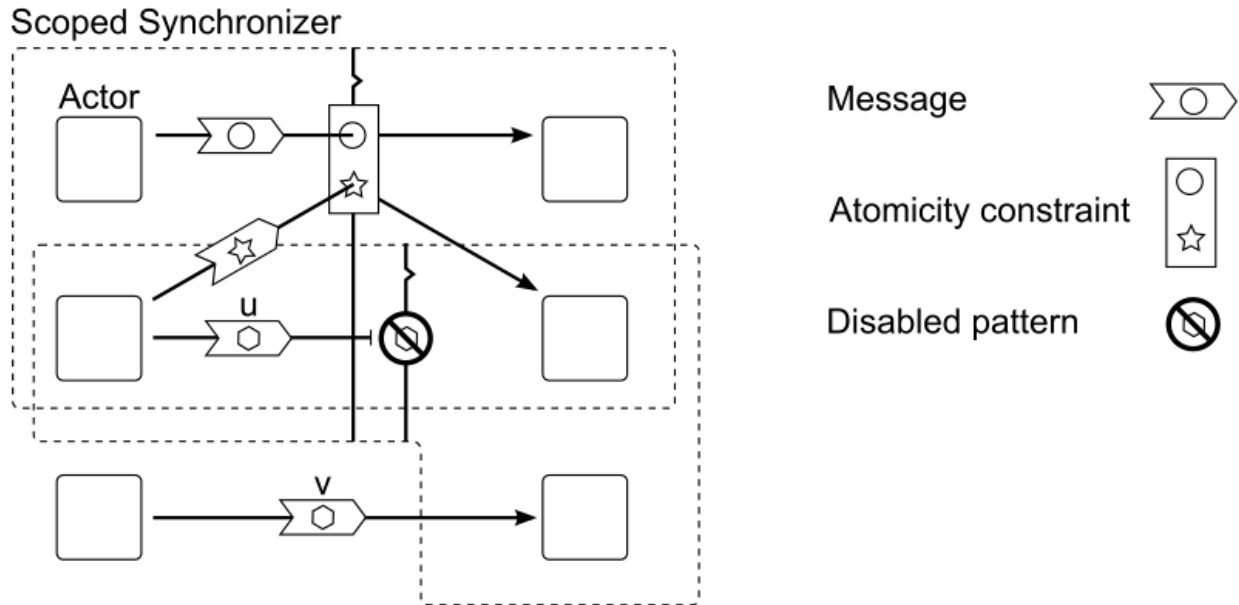


Figure 6. Constraints Enforced by Scoped Synchronizers

Scoped synchronizers (dashed frames) constrain only messages sent by actors for which they hold the synchronization-capability. These actors are placed in the left part of the synchronizer. Their sent messages must satisfy the constraints before they can be dispatched at the recipients (placed right). Since message u matches a disabling pattern of the lower synchronizer, it cannot be dispatched. However, the respective synchronizer lacks control over the sender of message v , so v can be dispatched despite having the same shape as u .

As with object-capabilities, we assume that synchronization-capabilities are unique across the system and cannot be guessed. Their distribution follows similar rules. Actors can obtain synchronization-capabilities through initialization and introduction. However, the parenthood rule is transitive: creating a new actor yields a synchronization-capability for this actor and all its children. The transitivity of synchronization-capabilities prevents actors from escaping synchronization constraints by transferring their behavior to a new actor, thereby changing their identity. Synchronization-constraints hence grant control over families of actors, including future members whose identities are yet unknown.

The two types of capabilities are separate; a capability of one type cannot be used in places that require the other. This separation allows actors to send messages to other, potentially untrusted actors, without submitting to the synchronization constraints of the recipient actors. In contrast to

the conventional synchronizer semantics, the semantics of scoped synchronizers ensures that the reply address contained inside a message can be used solely for communication.

With synchronizers only constraining messages for which they hold the synchronization-capabilities, it becomes unnecessary to restrict access to the synchronizer installation primitive. Any actor may therefore install synchronizers on all its acquaintances. The imposed constraints will simply stay without effect for most messages.

Synchronization-capabilities thus prevent the intentional deadlock scenarios discussed in section 3. In the *DisablingAttack* and *AtomicityAttack* synchronizer examples, scoping the situation is similar to that of the lower right actor in Figure 6: unless the synchronizers hold some relevant synchronization-capability, all messages will remain unaffected—as is the case for message *v* in the figure. Hence, the malicious installing actor poses no threat if none of the other actor in the system supplies it with a synchronization-capability. However, even in this case, the deadlock concerns only parts of the system. Synchronization-capabilities cannot completely prevent deadlocks that arise from incompatible constraints. Nevertheless, accidental interference of constraints becomes less likely.

Synchronization constraints determine whether a message can be dispatched (processed) at the receiving actor. Because communication is asynchronous, the sending actor cannot answer this question as the state of the recipient actor may change while the message is in transit. Synchronizers therefore reside at the receiving actors; they can be regarded as constraint servers that are queried by the message dispatch mechanism. This remains true despite the scoping mechanism's focus on message senders. The only change is that synchronizers now have to possess the right synchronization-capability to control a message.

An actor's scheduler can dispatch a message only if the message is not disabled by a synchronizer. The scheduler identifies applicable synchronizers by matching the message against the patterns declared by installed synchronizers. The scoped semantics requires not only that the pattern matches (as in conventional synchronizer semantics), but also that the synchronizer's synchronization-capability gives it control over the message.

When a message is dispatched, all synchronizers belonging to matching update patterns receive a notice. This includes synchronizers that lack the required synchronization-capability. Making the dispatch of messages public guarantees a consistent view on the system; it allows synchronizers to take into account the actions of the *uncontrolled* part of the environment. For example, consider the cooperating resource administrators above. If the *AllocationPolicy* synchronizer was blind to the requests and release messages of some users, then it could not enforce the intended limit on the total number of drive allocations on the users it controls.

However, a globally visible message dispatch is a trade-off. While it allows a consistent view on the system, it enables malicious actors to spy on other actors, as shown in Figure 7.

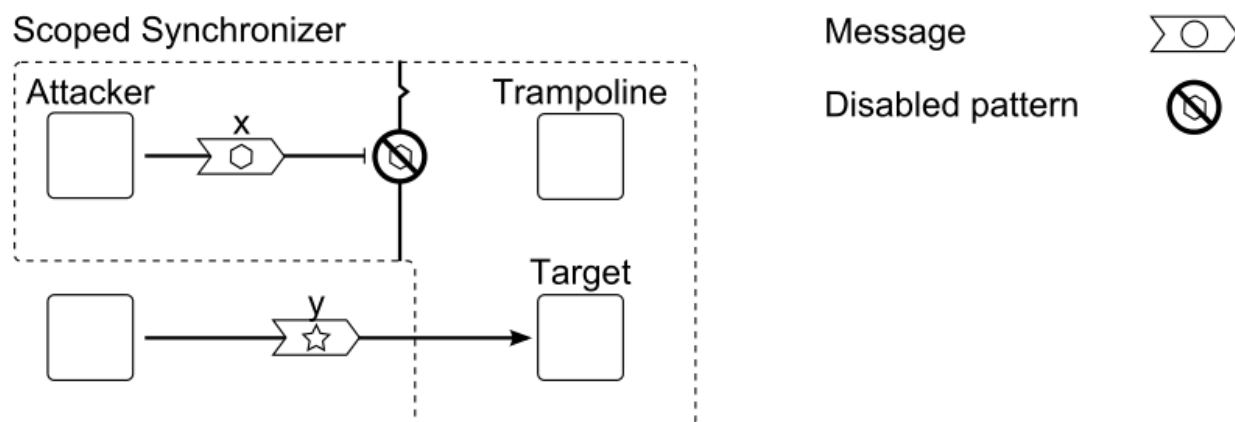


Figure 7. Information Leak through Updates

Scoping only limits the constraining power of Synchronizers. To guarantee a consistent view on the system, Synchronizers can observe all messages that an Actor dispatches—regardless of the synchronization-capabilities the synchronizer holds. The *Attacker* Actor exploits this fact to gather information about the *Target* Actor: First, the *Attacker* creates a *Trampoline* Actor and installs a Synchronizer on the *Target* and the *Trampoline*. The Synchronizer disables the dispatch of message *x* at the *Trampoline* until it observes message *y* at the *Target*. Then, the *Attacker* sends message *x* to the *Trampoline*. Once the *Trampoline* dispatches *x*, it bounces a message back to the *Attacker*, providing the *Attacker* with the knowledge that the *Target* dispatched message *y*.

Session Types for Actors

Session types are a means of expressing the order and type of messages exchanged by concurrently executing processes. In particular, session types can be used to statically check if a group of processes communicates according to a given specification. In these systems, a *global type* specifies the permissible sequences of messages that participants may exchange in a given *session*, as well as the types of these messages. The typing requires the programmer to provide the *global type*. A *projection* algorithm then generates the restrictions implied by the global type for each participant. Such restrictions are called *end-point types* or *local types* and describe the expected behavior of the individual participants in the protocol. The actual program code implementing the behavior of a participant is checked for conformance against this localized behavior specification. Conventional session types can be generalized to typing coordination constraints in *parameterized* actor programs, which can then be enforced using, e.g., *synchronizers*.

Typing coordination constraints in actors requires addressing two problems. First, asynchronous communication leads to delays that require considering arbitrary shuffles. Second, *parameterized* protocols must be considered. For example, assume two actors communicating through a sliding

window protocol: the actors agree on the length of the window (i.e., the number of messages that may be buffered) and then proceed to a concurrent exchange of messages. Conventional session types are not suitable for typing interactions such as the sliding window protocol. The reason for this limitation is that their respective type languages depend on other formalisms for type checking (such as typed λ -calculus or System T), and these formalisms do not support a concurrency construct.

We have developed a programming language, Lang-A, along with a session type system, System-A, that overcomes many of the aforementioned limitations through the use of novel constructs; in particular, the introduction of *parameterized* constructs for expressing asynchrony, concurrency, sequence, choice and atomicity in protocols, an inference algorithm that derives local System-A types from Lang-A programs, and a formal treatment of the type system.

Global Types

A global type describes a protocol to which the whole system must adhere. The sliding window protocol specification is a global type since it describes the behavior of all participants. Table 1 presents the grammar that generates the syntactic category G of global types. The elements of G , instances of global types, will be denoted by variations of the variable G . Intuitively, the rules capture the following concepts:

(G-Interaction) denotes the sending and receiving of a message.

(G-Seq) is used for the sequential composition of events.

(G-Choice) denotes exclusive choice between the arguments.

(G-Paral) means that the arguments run concurrently. Interleavings are allowed, as long as the order established by the $;$ operator is respected.

(G-Shuffle) means that both arguments are executed atomically, in an unspecified order. Formally, $G_1 \otimes G_2 \equiv (G_1 ; G_2) \oplus (G_2 ; G_1)$ with the \equiv relation denoting semantic equivalence.

(G-KleeneStar) has the usual semantics of zero or more repetitions of the argument. We assume a finite number of repetitions.

Table 1. The syntax of global types. The auxiliary symbols appearing in the grammar have the following domains: $i \in \text{IndexNames}$; $n, n_1, n_2 \in \text{ParamNames} \cup \mathbb{N}$; $a, b \in \text{ActorNames} \cup \{\alpha_j \mid \alpha \in \text{ActorNames}, j \in \text{IndexNames}\}$; and $m \in \text{MsgNames} \cup \{\mu_j \mid \mu \in \text{MsgNames}, j \in \text{IndexNames}\}$.

$\mathcal{G} ::= a \xrightarrow{m} b$	(G-Interaction)		(\mathcal{G})	(G-Paren)
$\mathcal{G} ; \mathcal{G}$	(G-Seq)		$\bigodot_{i=n_1}^{n_2} \mathcal{G}_i$	(G-Seq-N)
$\mathcal{G} \oplus \mathcal{G}$	(G-Choice)		$\bigoplus_{i=n_1}^{n_2} \mathcal{G}_i$	(G-Choice-N)
$\mathcal{G} \parallel \mathcal{G}$	(G-Paral)		$\parallel_{i=n_1}^{n_2} \mathcal{G}_i$	(G-Paral-N)
$\mathcal{G} \otimes \mathcal{G}$	(G-Shuffle)		$\bigotimes_{i=n_1}^{n_2} \mathcal{G}_i$	(G-Shuffle-N)
\mathcal{G}^n	(G-Exp)		\mathcal{G}^*	(G-KleeneStar)

The n -ary versions of the operators express behaviors where the value of n , n_1 , and n_2 are unknown at compile time. Intuitively, the rules (G-Seq-N), (G-Choice-N), (G-Parallel-N), and (G-Shuffle-N) apply the respective binary operator $n_2 - n_1 + 1$ times, generating a global type for each of the $n_2 - n_1 + 1$ values of i . (G-Exp) denotes the n -fold, sequential repetition of the argument. Note that for known parameter values, these expansions can take place during compilation.

All of the operators are commutative, with the exception of sequencing. All operators are furthermore associative, with the exception of shuffling. In particular,

$$\bigotimes_{i=1}^n G_i \neq (\dots ((G_1 \otimes G_2) \otimes G_3 \dots) \otimes \dots \otimes G_n)$$

because the meaning is that all arguments G_i are executed atomically, but in an unspecified order. Instead, the right-hand side above prevents, for example, G_3 from occurring between G_1 and G_2 .

The distinction between the Kleene star and exponentiation is fundamental. The use of G^n means that the protocol conformance checker will have to prove that the system is correct for any fixed value of the parameter n . G^* on the other hand means an unbounded number of repetitions of G . There is no parameter fixing this number, and it may be different from instance to instance of the Kleene star and/or among executions of the same program with the same run-time values for its parameters. The Kleene star entails a choice as to when to exit the loop.

The traces of a global type $G \in \mathcal{G}$ capture the permissible sequences of messages that participants may exchange. More formally, an *event* is defined as a single interaction $p_1 \xrightarrow{m} p_2$. A trace is a finite sequence of events and is of the form $e_1 ; e_2 ; \dots ; e_k$.

Programming Language

Global types by themselves provide no implementation of protocols; implementations are given in the language Lang-A. A Lang-A program begins with declaring the program parameters, akin to System-A parameters. Then come message structure definitions, and the code for each actor. Both actor and message definitions can include an optional array syntax after their name. In the case of actors, this syntax declares as many of them as the array parameter. In the case of message structures, it declares as many *message types* as the array parameter. This allows the expression of protocols where both actor names and message types are parameterized. Lang-A is defined so that there is almost a one-to-one correspondence between the language constructs and the syntax of local types, described below.

Figure 8 shows an implementation in Lang-A of the sliding window protocol. The *spawn* statement launches n parallel instances of its block argument, one for each value of the provided index expression. Sends and receives coming from different spawned operations can be interleaved in any way possible. In this example, both the sender and the receiver spawn n parallel operations, each consisting of a repeating send/receive pair. This allows any interleaving of sends and receives, as long as no more than n sends are left unacknowledged.

```

n : param

// the sender
actor a = {
  message m : Int
  message ack : Int
  var NotDone : Boolean
  NotDone = true

  spawn(i = 1..n
    while NotDone {
      m = ...
      send(b, m) ;
      rcv(b, ack) ;
      NotDone = ...
    })
}

// the receiver
actor b = {
  message m : Int
  message ack : Int
  var NotDone : Boolean
  NotDone = true

  spawn(i = 1..n
    while NotDone {
      rcv(a, m) ;
      ack = ...
      send(a, ack) ;
      NotDone = ...
    })
}

```

Figure 8. The sliding window protocol in Lang-A

Local Types

A local type specifies the abstract behavior of a single protocol participant, for example of one of the actors in a Lang-A program. Furthermore, local types specify the behavior restrictions that a global type implies for each protocol participant. The main use of local types, which can be inferred from Lang-A syntax, is to check whether a program conforms to a global type.

The syntactic category L of local types is defined by the grammar in Table 2. We will use the variable $L \in L$, often indexed, to refer to local types. Besides characterizing actor behavior, local types also specify the behavior restrictions that a global type implies for each participant. In the grammar,

(L-Send) denotes sending a message of type t to actor a .

(L-Recv) denotes receiving a message of type t from actor a .

(L-Seq), **(L-Choice)**, **(L-Shuffle)**, **(L-Exp)**, **(L-KleeneStar)** describe the same concepts as in the global types.

(L-Paral) is also defined as in the case of global types.

Table 2. The syntax of local types. As in the syntax of global types, the grammar contains the auxiliary symbols.

$\mathcal{L} ::=$	(\mathcal{L})	(L-Paren)	τ	(L-Empty)
	$ a!t$	(L-Send)	$ a?t$	(L-Recv)
	$ \mathcal{L} ; \mathcal{L}$	(L-Seq)	$ \bigodot_{i=n_1}^{n_2} \mathcal{L}_i$	(L-Seq-N)
	$ \mathcal{L} \oplus \mathcal{L}$	(L-Choice)	$ \bigoplus_{i=n_1}^{n_2} \mathcal{L}_i$	(L-Choice-N)
	$ \mathcal{L} \parallel \mathcal{L}$	(L-Paral)	$ \parallel_{i=n_1}^{n_2} \mathcal{L}_i$	(L-Paral-N)
	$ \mathcal{L} \otimes \mathcal{L}$	(L-Shuffle)	$ \bigotimes_{i=n_1}^{n_2} \mathcal{L}_i$	(L-Shuffle-N)
	$ \mathcal{L}^n$	(L-Exp)	$ \mathcal{L}^*$	(L-KleeneStar)

Type Checking

After inferring the local types in a Lang-A program, and projecting out the local types from a global type for all participants, we can check a program's conformance to a given global type. To perform the comparison, local types are reduced to a *normal form*. Types in normal form are

simpler in several ways, e.g., complex operator applications have been replaced simpler ones, and the terms of commutative operators have been rearranged to a deterministic order.

Due to how normalization is defined, for any local type $L \in \mathcal{L}$ in System-A, there exists a finite sequence of reduction steps which brings the type to a normal form. In addition, a type L and its normal form L_{norm} are trace equivalent.

Realization of Global Types

A global type must satisfy certain properties in order to be *projectable*. Applying the projection function to a projectable global type will result in local types for the participants whose combined behavior is consistent with the global type.

First, to be projectable, the sequential constructs of a global type must retain their sequential semantics after projection. In addition, it must be ensured that projecting $G_1 \oplus G_2$ maintains the choice semantics, meaning that all participants can recognize which branch of the choice operator they need to take during execution. For shuffling to be projectable, it has to be the case that the constituents can be sequenced both ways, and also that the right hand side satisfies the choice projectability criteria. In general, the problem with the shuffle operator appears when actions in one concurrent branch affect choices made on another. Global types that do not exhibit this problem are *concurrently projectable*. Use of the Kleene star in global types can result in protocols whose projection is unsafe, that is, can result in execution traces that are not part of the original global type. To avoid this, a global type must be such that the entry and exit conditions to the starred type can be identified by all participants.

Correctness

If a global type G is projectable according to the criteria sketched above, the projection function generates local types which are functionally consistent with the global type. We denote the environment resulting from the projection of G onto each one of the participants by Δ_G . That is, $\Delta_G = \{ p : G \rightarrow p \mid p \in \Pi \}$ where Π is the set of participating actors. The set of traces $tr(\Delta)$ producible by an environment Δ is the union of the sets of traces producible by the local types in Δ . Now, let PR be the set of projectable global types. The key correctness property is then that $G \in PR \Rightarrow tr(G) = tr(\Delta_G)$. The proof is by induction on the structure of global types.

Section 4 Results and Discussion

IMCM Framework

We measure effectiveness as the speedup gained compared to sequential local execution on mobile device in order to make the results comparable and link them to our target offloading goal of maximizing application performance. Our selected corpus consists of applications covering different types of programs: CPU intensive, communication intensive, I/O intensive, and combined. We investigate the effect of different application parameters on offloading decision and evaluate the performance of the IMCM middleware framework.

Experimental Setup

The equipment we used include a Samsung Google Nexus S as the mobile device and a Macbook Pro Laptop as the remote offloading server. Table 3 summarizes the specifications of our used equipment. Mobile device and the remote server are both on the same WiFi network.

Table 3. Specifications of the used equipment for evaluation

	Remote Server	Mobile Device
System	Macbook Pro-Retina	Samsung Google Nexus S
OS	Mac OSX 10.9.4	Android 4.1.2
VM	JVM (JRE 1.6)	DalvikVM
Processor	Intel Core i7	ARM Coretex-A8
Proc. speed	2.3 GHz	1 GHz
No. of cores	4	1
L2 Cache	256 KB/Core	256 KB
L3 Cache	6MB	-
Memory	16 GB	512 MB

Our mobile-cloud application model is based on the actor model of computing that offers natural parallelism for developed applications. Many actor programming languages have been developed over years to support different applications. Despite some small differences, most of these programming languages provide main standard actor semantics including *encapsulation*, *fair scheduling*, *location transparency*, *locality of references*, and *transparent migration*. For our

experiments, we chose SALSA as the programming language mainly due to its adherence to standard actor semantics. SALSA provides good support for parallel and distributed programming. Its support for code and data mobility and asynchronous message passing makes programming for distributed systems a natural task. Its coordination model provides an attractive feature for parallel programming where multiple CPUs need to coordinate and communicate between themselves in an efficient manner. SALSA depends on Java, hence it inherits Java's powerful feature of portability across different platforms. We were able to make SALSA work on Android mobile devices running DalvikVM with some modifications. SALSA provides lightweight actors. The use of lightweight actors makes SALSA highly scalable that is one of the main limitations of some older actor languages. A huge advantage of using lightweight actors is the speed and ease of actor migration between different devices. Our experimental result showed that SALSA actors are usually small in size (if not carrying large amount of internal data) and can be created or migrated in 100 ~ 200 ms on or between different machines working on the same WiFi network. This fast migration speed eases the process of mobile-cloud application offloading.

The base case in our evaluation is the required time for local sequential execution of the application on the mobile device and the execution speedups are used for comparing different scenarios. In order to account for randomness, we repeat each experiment five times and verify the statistical significance of observed execution times through non-parametric Mann-Whitney U-tests. Unless stated otherwise, the test is two-tailed and the significance level is $\alpha = 0.01$.

Program Corpus

Table 4 lists the programs used in the evaluation together with their main characteristics. Evaluation benchmark programs are selected based on their characteristics to cover different application behavior: Computational intensive, Communication intensive, and I/O intensive. In addition, a multi-behavior application is added to combine different characteristics. To avoid a bias towards specific strengths of our approach and to foster comparability, we mostly use similar examples as for works presenting solutions to mobile-cloud computation offloading. The *NQueen* program is a computation-intensive application that places N queens on a $N*N$ chessboard so that no two queens threaten each other. The *Heat* program is a communication-intensive application that simulates heat transfer in a two-dimensional grid in an iterative fashion. Our implementation allows specifying the desired level of communication and both medium and high level of communications are studied. The *Trap* program is a computation-intensive application that calculates a definite integral by approximating the region under the graph as a trapezoid and calculating its area. The *Virus* program reads in file streams from disk and scans for the signature of a given virus. The *Rotate* program is an I/O-intensive application that reads in an image from disk, rotates it in memory and writes it back to disk. Similarly, the *ExSort* program is an I/O-intensive application that sorts the content of a large file using external sort algorithm in limited amount of memory. Finally, the *Image* program combines all I/O, CPU, and communication characteristics by detecting and recognizing all faces in a given picture using

a large dataset of known faces. Since processing of each picture is performed sequentially, multiple images are processed simultaneously in order to add parallelism.

Table 4. Programs used to evaluate our framework. Application characteristic column shows dominant behavior of the application, raw speedup column summarizes maximum speedup gained by running application on a more-resourceful machine excluding offloading time, and offload speedup shows maximum speedup resulting from offloading including offloading overhead.

Experiment	Description	Application Characteristic				Raw Speedup
		Comp.	Comm.	I/O		
				read	write	
NQueen	Places N Queens on N*N board	intensive	-	-	-	73
Image	Detects & recognizes all faces in a photo	intensive	limited	limited	-	91
Trap	Uses trapezoidal rule to calculate definite integral	intensive	limited	-	-	30
Virus	Scans a file stream for a specific virus signature	-	-	intensive	-	28
Rotate	Reads, rotates & saves an image to disk	-	-	intensive	intensive	28
ExSort	External Sort of the content of a file	intensive	-	intensive	intensive	46
Heat1	simulates heat exchange on a board	limited	medium	-	-	31
Heat2	simulates heat exchange on a board	limited	high	-	-	14

Influence of Application Parameters on Offloading Decision

This section discusses how different application or execution properties influence offloading decision, answering the following research questions:

RQ1: What influence do the a) cost of offloading process, b) application type, and c) run-time parameters have on the mobile-cloud offloading decision?

Table 4 shows the speedup results for different applications together with applications' main characteristics. While the *raw speedup* column ignores the cost of offloading process, *offload speedup* column shows a more realistic view on mobile-cloud offloading by including the required time for offloading process. Note that different rows of the table relate to different applications with significantly different behavior, architecture and characteristics that should not be compared with each other. Comparing the values of *raw speedup* and *offload speedup* columns shows the effect of offloading cost on gained speedup. Offloading cost includes the required resources to make offloading decision, offload the application code to remote server and bringing back the result. Ignoring the cost of offloading process, The equation predicts the speedup resulting from running the same code on a faster machine. Assuming $X_{server} = 7$ for our experimental setup, the expected speedup is as below:

$$Speedup = \frac{S_s}{S_m} = \frac{2.3 * 4 * 7}{1.0 * 1} = 64 \quad (7)$$

raw speedup column shows that a speedup of 64 times or even higher is possible. However, when large amount of data needs to be offloaded (such as *Rotate* application), offloading speedup reached in practice is significantly lower. Moreover, the result highly depends on the application type and behavior as well. A computational-intensive application with high degree of parallelism (e.g. *NQueen*) can benefit from all the additional available resources on the remote server and can reach a high offloading speedup. Extensive I/O operations or communications between different components limits application's ability of benefiting from additional available computational resources at the remote server and reduces the gained speedup (e.g. *Rotate* and *Heat*).

In order to decide on the beneficiary of offloading w amount of computation to a remote server for our experimental setup, Equation 7 can be used with values from Table 3:

$$w * \left(\frac{1}{1024MHz} - \frac{1}{2.3 * 1024MHz * 4 * 7} \right) > \frac{d_i}{B} \quad (8)$$

Rearranging the equation to compute B_{min} to be the minimum required bandwidth in order for offloading decision to reduce total application execution time. The equation depends on the ratio of d_i / w and can only be true when the ratio is small enough. In other words, application offloading is beneficial for large amount of computation (w) and low amount of transferred data (d_i). For values in between, the decision depends on the available bandwidth (B) and an elasticity manager must evaluate the equation based on run-time parameters. For the *NQueen* problem, a single integer value has to be transferred both for input value (N) and final result and d_i is very small. At the same time, problem is computational-intensive and requires large amount of computation (large w). As a result, any type of network connection provides enough bandwidth and offloading always improves application performance. Note that the code of the *NQueen* solver is assumed to be available on the remote server and network latency is ignored. In case of the *Image*, assuming remote server to be very fast ($S_s = \infty$), offloading decision depends on w , d_i and B . If detection of faces in the initial image, extracting features for every detected face and comparison to database are all offloaded, the entire initial image needs to be transferred to the remote server and the amount of communicated data (d_i) is large. Thus, it is only beneficial to offload, if B is large enough. On the other hand, if the initial detection of faces are performed locally and only the extracted features are transferred, d_i is much smaller. Consequently, even for slower network connections, offloading of the remaining parts is beneficial. This highlights the importance of considering the combination of all parameters for deciding on offloading. Different parts of an application can become offloading candidates at different time and an elasticity manager is required to dynamically decide on offloading based on run-time parameters.

RQ2: How significant is the influence of problem size (amount of work) on mobile-cloud offloading?

Figure 9 and Figure 10 show the offloading speedup for different amount of work for *NQueen* and *Image* applications. The results show that larger amount of work results in more computationally-intensive applications, reduces the importance of the fixed amount of work required for offloading process, and increases the gained speedup. While initial offloading speedup of *NQueen* problem is almost equal to 1 (for $N = 8$) due to low amount of required computation, changing N value exponentially increases the amount of work to be performed and the resulting speedup. *Image* problem is a multi-behavior application with initial speedup of larger than 1 due to the size of computations required for processing even one single image. For this problem, changing the amount of work equals increasing the number of images to be processed and results in linear increase of speedup.

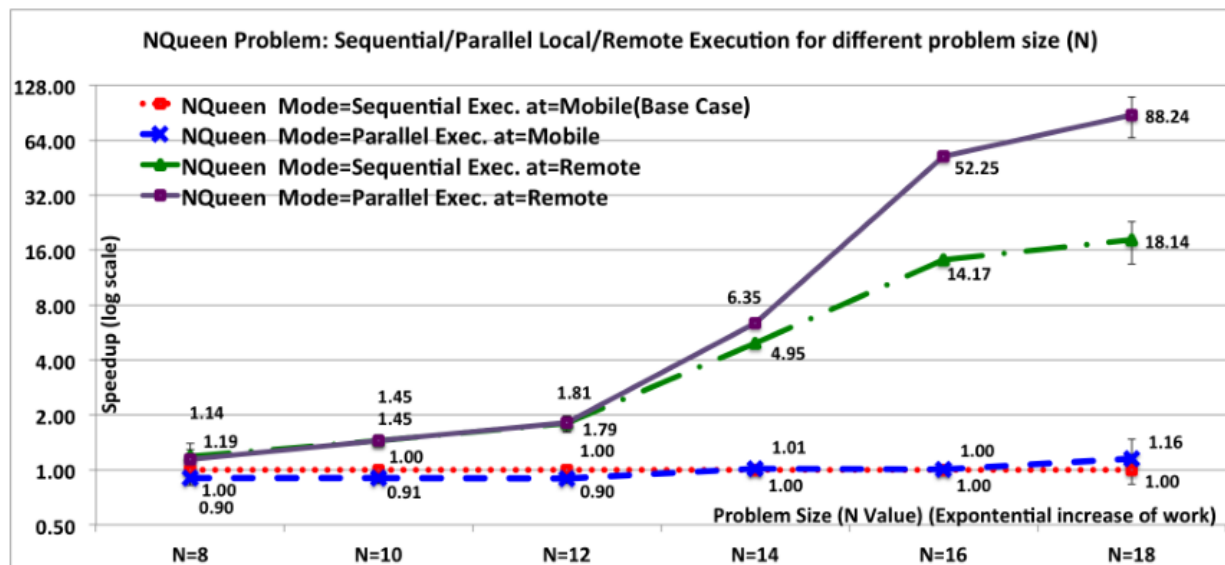


Figure 9. Speedup summary for local and remote execution of N-Queen execution for different amount of work (different problem size)

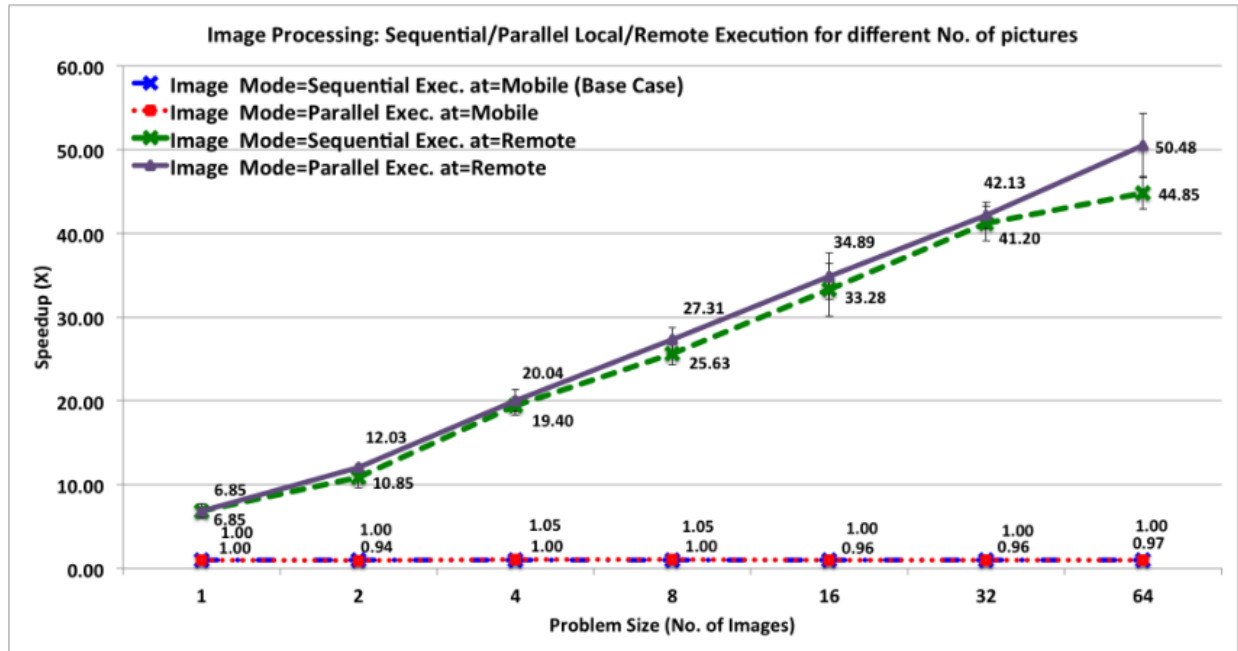


Figure 10. Speedup summary for local and remote execution of Image Processing application for different amount of work (different no. of images to process).

RQ3: What influence does the application parallelism degree have on mobile-cloud offloading?

If the ideal speedup resulting from offloading where computation is large enough, code has high degree of parallelism roughly comparable to available resources, and negligible amount of resources is used for offloading process. Without benefiting from parallelism, running the same code on a more resourceful machine can only provide limited speedup (sequential remote execution graphs of Figure 9 and Figure 10). This speedup is mostly because of benefiting from remote server's faster CPU speed, additional available caches, and more memory. However, additional available processing units are not used. We mentioned that for practical applications, the amount of resources required for offloading process is negligible compared to resources required for performing large amount of computation. If computation is not large enough, even using high degree of parallelism does not provide significant additional speedup. However, when the amount of computation is large enough, higher degree of parallelism significantly improves the performance and the benefit of having additional processing resources becomes visible.

Figure 11 shows the relationship between application parallelism degree and speedup resulting from offloading. While on a mobile device with only one single core increasing parallelism degree does not improve the performance, on a more resourceful remote server increasing the program parallelism degree allows better utilization of resources and increases application performance. While sequential execution of the *NQueen* problem on a faster system generates a speedup of 14 times, increasing the parallelism degree increases the resulting speedup to 55. Performance improvement resulting from increasing program parallelism degree is limited by the availability of resources. At a certain parallelism degree, resources will become saturated and

further increase of parallelism degree will have reverse negative effect (Figure 10). Considering the null hypothesis that remote sequential execution is as effective as the remote parallel execution, Mann-Whitney U-test shows that all differences for various problem sizes and parallelism degrees are significant. Consequently, the null hypothesis is rejected.

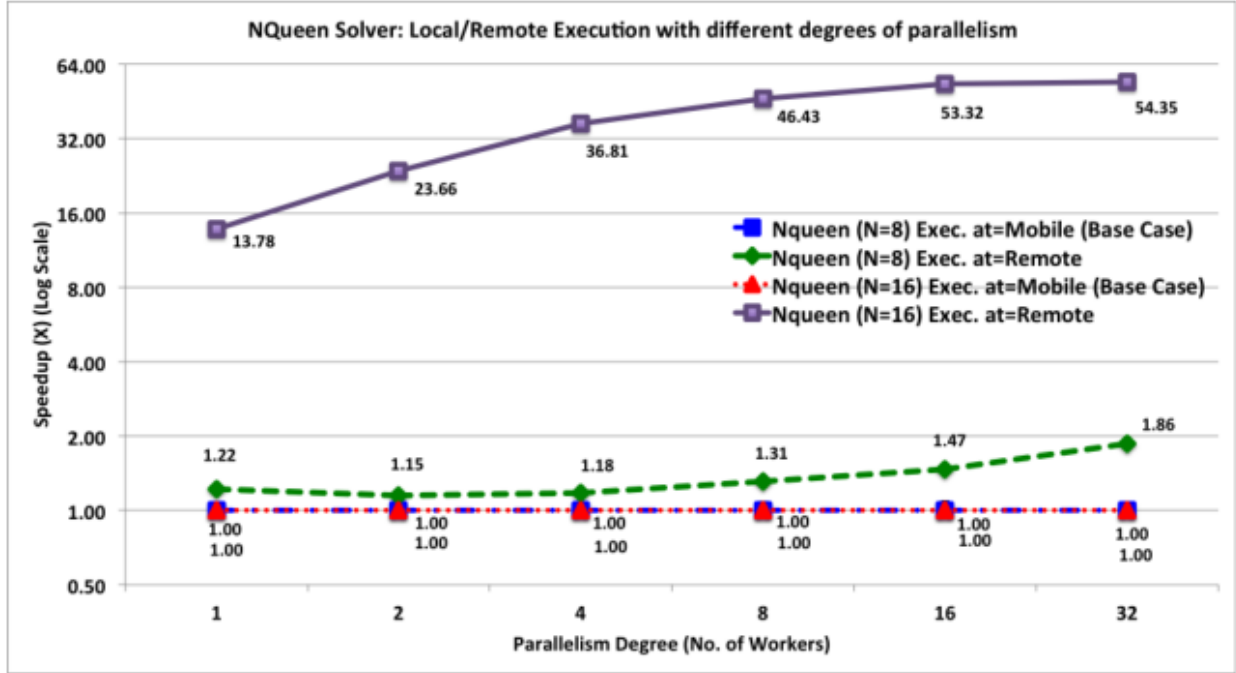


Figure 11. Speedup summary for local and remote execution of the *NQueen* problem with different degree of parallelism

Effectiveness of the Proposed Middleware Framework

This section discusses the performance of the proposed IMCM middleware framework, answering the following research questions:

RQ4: Is the IMCM proposed parallel local & remote execution offloading solution more effective than existing sequential (or pseudo-parallel) execution offloading solutions?

While offloading computation to a more resourceful system can improve overall application performance, mobile device local resources are wasted while waiting in the idle state for the result of offloaded code to be returned. With mobile devices becoming more powerful, this wasted computational power can be put to a better use. Our proposed framework supports simultaneous local and remote application execution and uses local mobile resources to execute other parts of an application while waiting for the offloaded code result.

Figure 12 shows the speedup differences between processing different number of images using only remote server and simultaneous execution on both local device and remote server. Since processing of a single image is sequential, for small amount of work (small number of pictures to

process), total execution time will be dominated by the required time for local mobile device to process its share. This will result in remote server starvation and waste of resources, as there will be no more job for it to process. However, with increase in the amount of work, there will always be enough job for remote server to perform and the advantage of using both local and remote server for application code execution becomes visible. Figure 13 shows the same effect based on application parallelism degree. We mentioned earlier that higher degree of parallelism will increase the flexibility of the application and results in higher offloading speedup. However, this is only true, if enough computational resources are available. As can be seen in the graph, increasing the parallelism degree (number of workers) initially results in higher speedup but after a certain point this effect is reversed. In fact, having higher degree of parallelism than the available resources results in over-saturation of resources, adds the overhead of managing all those workers, and reduces overall speedup. Our results show that required parallelism degree for an application to reach highest speedup is proportional to number of processing cores available. The coverage differences of any two different number of workers for both remote and simultaneous local and remote executions are significant $\alpha = 0.01$. Thus, the null hypothesis that there is no significant difference between image processing execution with different number of workers can be rejected.

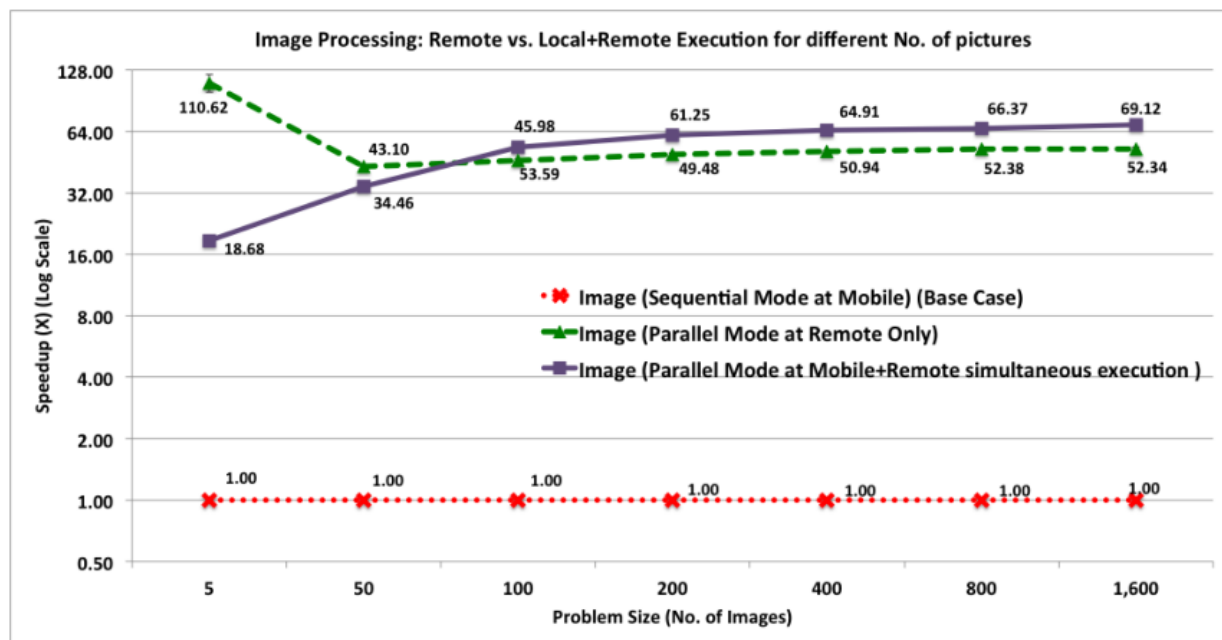


Figure 12. Speedup summary for remote execution vs. local + remote execution of image processing problem with different problem size (different number of images).

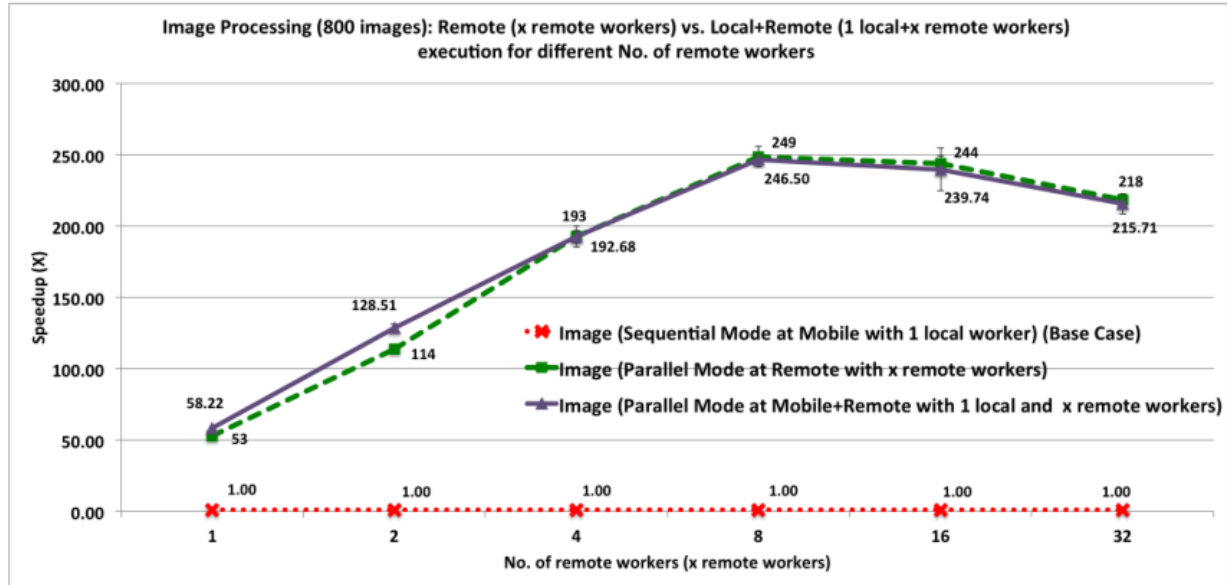


Figure 13. Speedup summary for remote execution (x remote workers) vs. local + remote execution (1 local + x remote workers) of image processing problem with different number of remote workers

RQ5: How effective is the IMCM elasticity manager in detecting application run-time environmental parameters and offloading appropriate application components?

Despite significant performance speedup resulting from offloading application to more resourceful systems, manual configuration of components between local mobile device and remote server is not possible. Ideal component distribution depends on several factors that can dynamically change during execution. Thus, an elasticity manager is required to monitor environmental changes and find optimal offloading plan. Figure 14 shows the result for manual placement of application components versus automatic component management using IMCM elasticity manager. Implemented elasticity manager uses the previous profiled execution times of different components at various locations to find the optimal location for placing every component for next interval. We currently do not use profiled execution time from previous execution of the application. As a result, there is an initial lag between start of an application and optimal placement of components resulting from the required time to collect enough profiled data. As a result, when problem size and resulting total application execution time increases, the gap between ideal placement of component and automatic distribution becomes narrower.

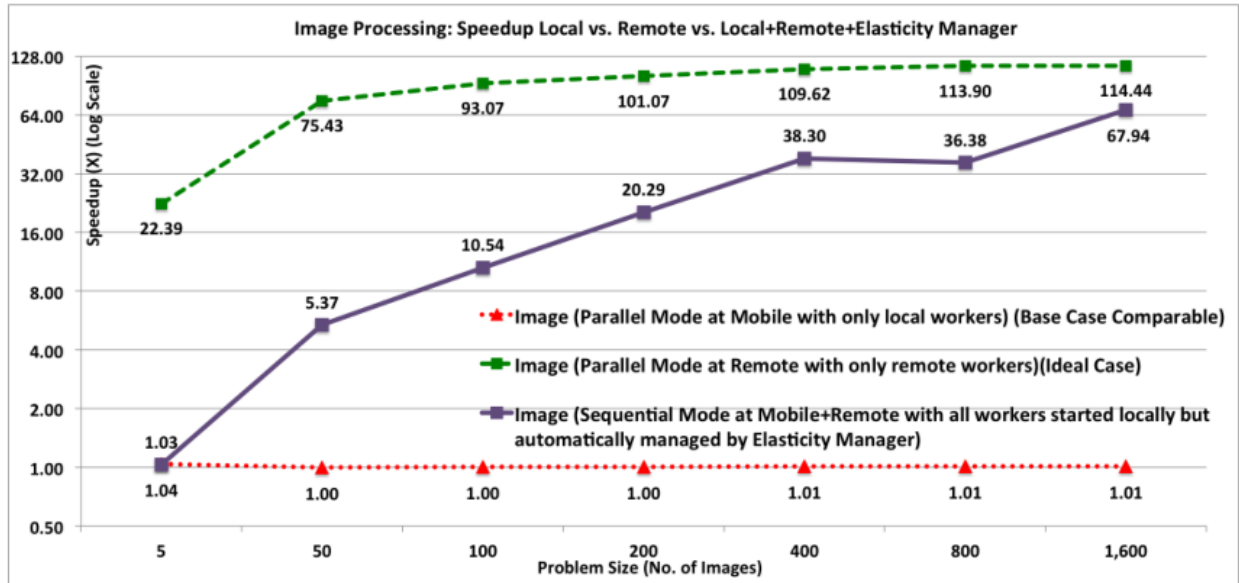


Figure 14. Speedup summary for local execution (base case) vs. remote execution (ideal case) vs. local execution with elasticity manager (all automatic management) of image processing problem with different problem size (different number of images to process)

RQ6: What is the performance overhead of the IMCM automatic elasticity manager?

While offloading appropriate components to a remote server can potentially improve application performance, having a costly elasticity manager to profile run-time and application parameters and finding optimal distribution plan can result in less overall performance. Figure 15 shows the overhead results from our implemented elasticity manager. Results show that having profiler and elasticity manager running in the background generates 1 - 5% speedup decrease on average. Considering the range of 9 - 60x for speedup gain from offloading applications shows that IMCM elasticity manager overhead is low. Moreover, as the problem size increases, the benefit of offloading becomes more dominant and the elasticity manager overhead becomes even less important.

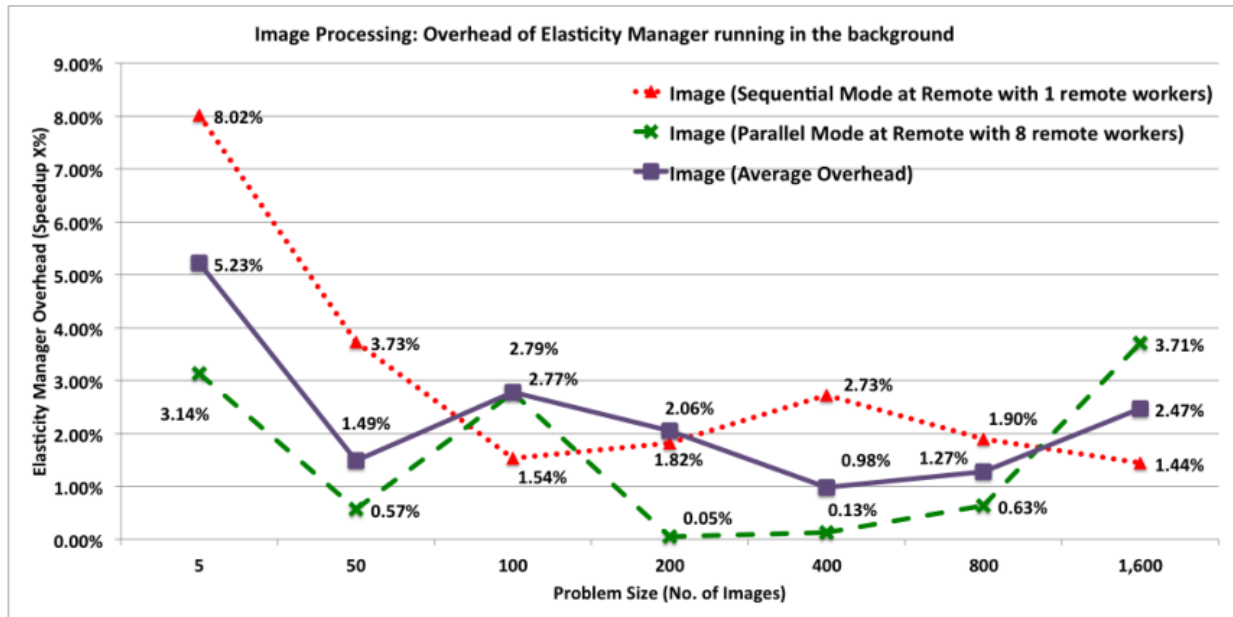


Figure 15. Overhead resulting from elasticity manager for image processing problem with different problem size (different number of images to process)

Synchronizers

In their current form, synchronizers are limited in their expressiveness through their choice to offer but a functional core consisting of two constraint types. An interesting opportunity for future research is extending the selection of available constraints. For instance, syntactic sugar like ordering constraints allows programmers to express their intentions more naturally, and thus make fewer mistakes. Other concepts like non-interleaving of message sequences cannot be expressed at all.

Another opportunity concerns the robustness of synchronizers against network partitions and crash failures. Augmenting the semantics with failure detectors appears to be a promising approach. A further interesting direction is finding methods for handling information leaks.

Finally, implementing synchronizers in a modern actor framework and conducting a large case study would give interesting insights into the (programmer and computational) performance of synchronizers.

Session Types

Adding support for dynamic process creation is an important direction for future work in session types for actor systems. In its current form, System-A cannot express actor creation as a behavior, and global types assume that all participants already exist. Matching a created actor with its subsequent use in a type requires an extra step which is not obvious.

Furthermore, System-A omits support for session delegation, and does not deal with issues of progress. In addition, it does not consider overlapping indexed names when nested in multiple operators. This disallows some cases, such as all-to-all communication.

Finally, other possible extensions concern the runtime monitoring application domain. In particular, adding support for global assertions can form the basis of a powerful theory for deriving local restrictions for each participant, which an asynchronous observer can then enforce.

Section 5 Conclusions

We have developed the IMCM middleware framework for transparent automatic code offloading from mobile devices to hybrid cloud spaces. The framework is fine-grained, supporting application configuration and distribution at the granularity of individual components; it is adaptive, addressing the dynamicity in run-time conditions and end-user contexts. It further supports component distribution in a hybrid cloud environment consisting of one or more public and private cloud spaces. Finally, it provides a new code offloading model that supports parallel program execution where application components located at mobile device and different cloud spaces are executed independently but concurrently. Evaluation results show that the offloading result depends on application behavior, offloading cost, and run-time parameters and can range between 9 to 56 times.

Future work based on the results of the project involves extending the framework to support mobile energy consumption optimization and to allow dynamic adjustment of application target goals. A big challenge with energy optimization is profiling detailed consumption of individual application components. While the execution time of different components can individually be recorded using the system clock, a mobile device only reports lump sum energy consumption, and obtaining a breakdown of total energy among different components remains as a challenge.

Section 3 Cyber Infrastructure Security: Dynamic Policy Monitoring with Interference in Cloud (Roy Campbell, John Bellessa, Shadi Noghabi, Luke Leslie, and Chris Cai)

Section 1 Summary of Research Project and Introduction

- Cyber Infrastructure Security: Dynamic Policy Monitoring with inference in cloud
- Large organizations' IT systems and critical infrastructure systems (e.g., airports, power grid) are system-of-systems composed of a large number of components vulnerable to attacks. Enforcing policies that cover different aspects of the system increases the overall system trustworthiness. The project aims at designing tools to enable monitoring/enforcing such policies in cloud computing infrastructure.

Section 2 Methods, Assumptions, and Procedures

Our approach permits each organization to monitor independently its own infrastructure, and exchange information with other organizations only when the local events are relevant to the detection of a violation. Our approach significantly reduces the amount of information shared while guaranteeing that all multi-organization policy violations can still be detected.

- Our approach uses the policies for determining which information should be shared. Through an analysis of the policies and of its current state, each organization independently decides which information should be shared, and with which organization, to guarantee the detection of violations.
- Our solution brings together the advantages of a distributed solution where each organization works independently to detect problems, with the advantages of a centralized solution that detects all inter-organization problems.

Section 3 Results and Discussion

Our approach permits each organization to monitor independently its own infrastructure, and exchange information with other organizations only when the local events are relevant to the detection of a violation. Our approach significantly reduces the amount of information shared while guaranteeing that all multi-organization policy violations can still be detected. It isolates virtual machines and helps prevent side channel attacks. In more detail:

- Our approach uses the policies for determining which information should be shared. Through an analysis of the policies and of its current state, each organization independently decides which information should be shared, and with which organization, to guarantee the detection of violations.
- Our solution brings together the advantages of a distributed solution where each organization works independently to detect problems, with the advantages of a centralized solution that detects all inter-organization problems.

- The approach leverages intrusion detection/VMI introspection, encryption, and software defined networks.

Section 4 Conclusions

Coupled with the results from Section 2, Containers/Software Defined Networking (SDN) - Security and Services, monitoring probes at the virtual machine level and preventing cross channel side attacks using the Intel SDX architecture, the research indicates that containment and isolation can be built for both virtual machines and containers. Multi-organizational policy-based monitoring and network traffic isolation through software defined network virtualization and hypervisors can be applied to cross security domain communication providing a new confidence in security within the Cloud.

Section 5 Recommendations

This research will continue, albeit at a slower pace without funding, to try to integrate these technologies into a practical solution.

Section 4 Design of Algorithms and Techniques for Real-time Assuredness in Cloud Computing (Indranil Gupta, PhD student: Mainak Ghosh; Graduated PhD students: Brian Cho and Imranul Hoque)

Section 1 Summary of Research Project

This part of the effort within the ACC project has designed, implemented, and experimentally evaluated new techniques that make today's cloud systems for storage, batch processing, and stream processing, more predictable in performance. This includes: i) support for priorities and real-time deadlines for Apache Hadoop jobs in YARN, ii) support for automated, background, and seamless reconfiguration operations in storage systems, and stream, and graph processing systems (Apache Cassandra, MongoDB, Apache Storm, LFGGraph), iii) support for consistency-latency SLAs/SLOs (Service Level Agreements/Objectives) for cloud storage systems (NoSQL databases; Apache Cassandra, Riak). Our work has contributed both algorithms and techniques, as well as incorporate modifications into open-source cloud systems including Hadoop, Cassandra, Riak, MongoDB, Apache Storm, etc.

Further, we are collaborating with co-PI Jose Meseguer in assuring predictability by using model-checking tools on several of these cloud systems, from NoSQL to transactional databases (model-checking work covered in a different section).

Section 2 Introduction

Cloud computing today relies heavily on distributed systems (typically open-source) running at large scales inside datacenters and under unpredictable failure-prone environments. Production deployments are subjected to workloads that are diverse and dynamic, have to deal with large numbers of machines and large amounts of data, and yet at the same time they need to meet service/application requirements for low latency, consistency, and high throughput.

This part of the effort within the ACC center has made contributions to making today's distributed systems more predictable even in dynamic environments. Predictability comes from the ability to support customer requirements such as deadlines and SLAs/SLOs (Service Level Agreements/Objectives), job deadlines and priorities, the ability to scale out/in seamlessly when requested by the admin, and to assure the system is up and running in spite of reconfiguration changes that may be occurring in the background. All of these problems are hot topics of discussion today and much needed by developers and deployers of large-scale cloud systems. We make contributions that are both theoretical and algorithmic, and also implement our techniques into several open-source systems.

Section 3 Methods, Assumptions, and Procedures

1. In the Natjam system [SoCC 2013] we support priorities and deadlines for jobs in Hadoop YARN. The key idea is to preempt the tasks of lower priority jobs so that resources can be freed up immediately for higher priority jobs, however we also checkpoint these preempted tasks so that they are work-conserving and such preempted tasks can resume from where they left off in the future (when resources become available for them). We explored policies for both job eviction (which jobs to victimize) and task eviction and found that evicting the job with most resources and the tasks with shortest time remaining is the best policy. Natjam was implemented into Apache Hadoop YARN.
2. In the Morplus system [ICAC 2015, IEEE TETC 2015, ICCAC 2015] we support reconfiguration operations for sharded NoSQL databases. Examples include changing the shard key, or scale out/in. The key ideas are to place the new shards at existing servers (we do not use new servers) using maximum matching so as to minimize network transfer volume, and to transfer the data in the background smartly, while still supporting reads and writes in the foreground. Morplus was implemented in MongoDB as well as Apache Cassandra.
3. In our two systems of Stela [IEEE IC2E 2016] and elastic graph processing [IEEE IC2E 2016] we support scale out and scale in (increase/decrease the number of machines/VMs) for stream processing systems and graph processing systems in such a way that the reconfiguration does not interrupt ongoing computation and gives close to optimal performance after the scaling. The key ideas in Stela, for instance, are to calculate which operators/tasks benefit from the most from the additional resources (on scale out) or are affected the least on shrinking (on scale in), and then to initiate such changes to them in the background. These systems were implemented in Apache Storm and LFGGraph respectively.
4. In the PCAP project [ACM TAAS 2017] we first extend the classical CAP theorem (which says that all 3 properties consistency, availability and partition-tolerance are impossible to achieve together) to be a probabilistic impossibility theorem that takes into account parameters for each of C, A, and P. This allows us to support latency SLAs/SLOs as well as consistency SLAs/SLOs for NoSQL databases, a sorely needed requirement in these otherwise-weakly-consistent databases. PCAP was integrated into both Apache Cassandra and Riak.

Section 4 Results and Discussion

All experiments were done with the production open-source systems, using traces from industry where available.

1. Natjam was evaluated with both microbenchmarks and traces from Yahoo!'s Hadoop production cluster. Natjam comes within 2% of ideal runtime for low priority jobs and within 7% of idea for high priority jobs. It is significantly better than existing techniques of either killing low priority tasks (Capacity scheduler) or waiting for such tasks to finish. When evaluated on real traces from Yahoo!'s Hadoop clusters, 53-63% of jobs improve in their runtime, and there is minimal starvation of jobs.
2. Morphus was evaluated with the industry benchmark YCSB (Yahoo Cloud Serving Benchmark). Morphus is able to complete reconfigurations quickly using over 50% of network bandwidth. It leaves read and write latencies unchanged (in fact improves median write times slightly!) and causes a very small drop in availability (only for writes).
3. These systems were evaluated with microbenchmarks and with production Storm topologies from Yahoo!. Stela and elastic graph processing are able to scale out/in quickly and without affecting the ongoing computation – for graph processing our techniques incur only a 6-11% overhead compared to the ideal. For stream processing our Stela system converges about 75-85% faster than the base Apache Storm system. While the base Storm system sometimes degrades in performance after scale out, Stela gives proportionally higher performance when machines are added.
4. PCAP was evaluated using both the industry YCSB benchmark, as well as network delay models from literature. The PCAP system always satisfies SLOs in spite of highly dynamic scenarios. In terms of optimizing other metrics (which are not part of the SLO) PCAP performs very close to the optimal achievable boundary when network conditions are good to reasonable. PCAP requires only minimal modifications to the underlying NoSQL system. We also extended PCAP to geo-distributed settings.

Section 5 Conclusions

Our work shows that today's distributed systems for storage, batch processing and stream processing can be made predictable. Developers/deployers can specify *what* they need in a declarative way using SLAs/SLOs, deadlines, or priorities, and our techniques allow the system itself to decide *how* to achieve these requirements (today's state of the art requires developers to grapple with both the what and the how).

Ongoing Work: Our ongoing work is extending the scale out/in building blocks we have designed for graph processing and stream processing into systems that support SLAs/SLOs for multi-tenant clusters that have stream processing and graph processing jobs. A separate related

project is starting to look at distributed machine learning systems, and making them truly multi-tenant.

Section 6 Recommendations

Cloud systems should use increase the use of declarative ways of specifying requirements from users and developers. SLAs/SLOs should be standardized. Further work is needed on: i) extending the richness of these SLAs/SLOs while still keeping them user-facing and away from the innards of the system, and ii) extending the notion of such requirements to other emerging areas of distributed systems such as distributed machine learning (like TensorFlow).

Section 5 Greatly Increase the Assurance Level to Cloud Computing Systems through Formal Specifications and Verification in Maude (Jose Meseguer, Si Liu, Peter Olveczky, and Stephen Skeirik)

Section 1 Summary of Research Project

To deal with large amounts of data while offering high availability, throughput and low latency, cloud computing systems rely on distributed, partitioned, and replicated data stores. Such cloud storage systems are complex software artifacts that are very hard to design and analyze. Formal specification and model checking analysis can significantly improve their design and validation. In the ACC project we have rewriting logic and its accompanying Maude tools as a suitable framework for formally specifying and analyzing both the correctness and the performance of cloud storage systems. Specifically, we have used rewriting logic to model and analyze industrial cloud storage systems such as Google's Megastore, Apache Cassandra, Apache ZooKeeper, and RAMP.

Section 2 Introduction

Vision: Formal Methods for Cloud Storage Systems

Our vision is to use formal methods to design cloud storage systems and to provide high levels of assurance that their designs satisfy given correctness and performance requirements. In a formally-based system design and analysis methodology, a mathematical model S describes the system design at the appropriate level of abstraction. This system specification S should be complemented by a formal property specification P that describes mathematically (and therefore precisely) the requirements that the system S should satisfy. Being a mathematical object, the model S can be subjected to mathematical reasoning (preferably fully automated or at least machine-assisted) to guarantee that the design satisfies the properties P . If the mathematical description S is executable, then it can be immediately simulated; there is no need to generate an extra artifact for testing and verification. An executable model can also be subjected to various kinds of model checking analyses that automatically explore all possible system behaviors from a given initial system configuration. From a system developer's perspective, such model checking can be seen as a powerful debugging and testing method that can automatically find subtle "corner case" bugs and that automatically executes a comprehensive "test suite" for complex fault-tolerant systems. We advocate the use of formal methods throughout the design process to quickly and easily explore many design options and to validate designs as early as possible, since errors are increasingly costly the later in the development process they are discovered. Of course, one can also do a postmortem formal analysis of an existing system by defining a formal model of it in order to analyze the system formally; we have shown the usefulness of such postmortem analysis in the modeling and design of the Cassandra system.

Performance is as important as correctness for storage systems. Some formal frameworks provide probabilistic or statistical model checking that can give performance assurances with a given confidence level.

What properties should a formal framework have in order to be suitable for developing and analyzing cloud storage systems in an industrial setting? The requirements can be summarized as follows:

1. Expressive languages and powerful tools that can handle very large and complex distributed systems. Complex distributed systems at different levels of abstraction must be expressible without tedious workarounds of key concepts (such as, e.g., time and different forms of communication). This requirement also includes the ability to express and verify complex liveness properties. In addition to automatic methods that help users diagnose bugs, it is also desirable to be able to machine-check proofs of the most critical parts.
2. The method must be easy to learn, apply, and remember, and its tools must be easy to use. The method should have clean simple syntax and semantics, should avoid esoteric concepts, and should use just a few simple language constructs. The author also recommends against distorting the language to make it more accessible, as the effect would be to obscure what is really going on.
3. A single method should be effective for a wide range of problems, and should quickly give useful results with minimal training and reasonable effort. A single method should be useful for many kinds of problems and systems, including data modeling and concurrent algorithms.
4. Modeling and analyzing performance, since performance is almost as important as correctness in industry.

Section 3 Methods, Assumptions, and Procedures

The Rewriting Logic Framework

Satisfying the above requirements is a tall order. We have found that rewriting logic and its associated Maude tool is a suitable framework for formally specifying and analyzing cloud storage systems.

In rewriting logic, data types are defined by algebraic equational specifications. That is, we declare sorts and function symbols; some function symbols are constructors} used to define the values of the data type; the others denote defined functions that are defined in a functional programming style using equations. Transitions are then defined by rewrite rules of the form $l \Rightarrow$

r, where l and rare terms (possibly containing variables) representing local state patterns. Rewriting logic is particularly suitable for specifying distributed systems in an object-oriented way, in which case the states are multisets of objects and messages (traveling between the objects), and where an object o of class C may have attributes a1, ...,an. For example, a rewrite rule

$$\begin{aligned} \text{rl [l] : } & m(O,w) \\ & \langle O : C \mid a1 : x, a2 : O', a3 : z \rangle \\ \Rightarrow & \\ & \langle O : C \mid a1 : x + w, a2 : O', a3 : z \rangle \\ & m'(O',x) . \end{aligned}$$

\

defines a family of transitions in which a message m with parameters O and w is read and consumed by an object O of class C, the attribute a1 of the object O is changed to x + w, and a new message m'(O',x) is generated.

Maude is a specification language and high-performance simulation and model checking tool for rewriting logic. Simulations --which simulate single runs of the system---provide a first quick initial feedback of the design. Maude reachability analysis--which checks whether a certain (un)desired state pattern can be reached from the initial state---and linear temporal logic (LTL) model checking---which checks whether all possible behaviors from the initial state satisfy a given LTL formula---can be used to analyze all possible behaviors from a given initial configuration.

The Maude tool ecosystem also includes Real-Time Maude, which extends Maude to real-time systems, and probabilistic rewrite theories, a specification formalism for specifying distributed systems with probabilistic features. A fully probabilistic subset of such theories can be subjected to statistical model checking analysis using the PVeStA tool. Statistical model checking performs randomized simulations until a probabilistic query can be answered (or the value of an expression be estimated) with the desired statistical confidence.

Rewriting logic and Maude address the above requirements as follows:

1. Rewriting logic is an expressive logic in which a wide range of complex concurrent systems, with different forms of communication and at various levels of abstractions, can be modeled in a natural way. In addition, its real-time extension supports the modeling of real-time systems. The Maude tools have been applied to a range of industrial and state-of-the-art academic systems. Complex system requirements, including safety and liveness properties, can be specified in Maude using linear temporal logic, which seems to be the most intuitive and easy-to-understand

advanced property specification language for system designers. We can also define functions on states to express nontrivial reachability properties.

2. Equations and rewrite rules: these intuitive notions are all that have to be learned. In addition, object-oriented programming is a well-known programming paradigm, which means that Maude's simple model of concurrent objects should be attractive to designers. We have experienced in other projects that system developers find object-oriented Maude specifications easier to read and understand than their own use case descriptions, and that students with no previous formal methods background can easily model and analyze complex distributed systems in Maude. The Maude tools provide automatic ("push-button") reachability and temporal logic model checking analysis, and simulation for rapid prototyping.

3. As mentioned, this simple and intuitive formalism has been applied to a wide range of systems, and to all aspects of those systems. For example, data types are modeled as equational specification and dynamic behavior is modeled by rewrite rules. Maude simulations and model checking are easy to use and provide useful feedback automatically: Maude's search and LTL model checking provides a counterexample trace if the desired property does not hold.

4. We have shown in previous work that randomized Real-Time Maude simulations (e.g., of wireless sensor networks) can give performance estimates as good as those of domain-specific simulation tools. More importantly, we can analyze performance measures and provide performance estimations with given confidence levels using probabilistic rewrite theories and statistical model checking; e.g.: "I can claim with 90% confidence that at least 75% of the transactions satisfy the property P." For performance estimation for cloud storage systems see the results section.

To summarize, a formal executable specification in Maude or one of its extensions allows us to define a single artifact that is, simultaneously, a mathematically precise high-level description of the system design and an executable system model that can be used for rapid prototyping, extensive testing, correctness analysis, and performance estimation.

Section 4 Results and Discussion

We summarize below some of the key results obtained in the work performed at the Assured Cloud Computing Center at the University of Illinois at Urbana-Champaign using Maude and its extensions to formally specify and analyze the correctness and performance of several important industrial cloud storage systems and a state-of-the-art academic one. In particular, we can list the following contributions:

1. Apache Cassandra. This is a popular open-source industrial key-value data store that only guarantees eventual consistency. We were interested in: (i) evaluating a proposed variation of Cassandra, and (ii) analyzing under what circumstances---and how often in practice---Cassandra also provides stronger consistency guarantees, such as read-your-writes or strong consistency. After studying Cassandra's 345,000 lines of code, we first developed a 1,000-line Maude specification, which captured the main design choices. Standard model checking allowed us to analyze under what conditions Cassandra guarantees strong consistency. By modifying a single function in our Maude model we obtained a model of our proposed optimization. We subjected both of our models to statistical model checking using PVeStA; this analysis indicated that the proposed optimization did not improve Cassandra's performance. But how reliable are such formal performance estimates? To investigate this question, we modified the Cassandra code to obtain an implementation of the alternative design, and executed both the original Cassandra code and the new system on representative workloads. These experiments showed that PVeStA statistical model checking provides reliable performance estimates. To the best of our knowledge this was the first time that, for key-value stores, model checking results were checked against a real system deployment, especially on performance-related metrics.

2. Megastore. This is a key part of Google's celebrated cloud infrastructure. Megastore's trade-off between consistency and efficiency is to guarantee consistency only for transactions that access a single entity group. It is obviously interesting to study such a successful cloud storage system. Furthermore, one of us had an idea on how to extend Megastore so that it would also guarantee strong consistency for certain transactions accessing multiple entity groups without sacrificing performance. The first challenge was to develop a detailed formal model of Megastore from the short high-level description. We used Maude simulation and model checking throughout the formalization of this complex system until we obtained a model that satisfied all desired properties. This model also provided the first reasonable detailed public description of Megastore. We then developed a formal model of our extension, and estimated the performance of both systems using randomized simulations in Real-Time Maude; these simulations indicated that Megastore and our extension had about the same performance. (Note that such ad hoc randomized simulations do not give a precise level of confidence in the performance estimates.)

3. RAMP, is a state-of-the-art academic partitioned data store that provides efficient lightweight transactions that guarantee the simple ``read atomicity'' consistency property. The RAMP designers have given hand proofs of correctness properties and proposed a number of variations of RAMP without giving details. We used Maude to: (i) check whether RAMP indeed satisfies the guaranteed properties, and (ii) develop detailed specifications of the different variations of RAMP and check which properties they satisfy.

4. ZooKeeper is a fault-tolerant distributed key/value data store that provides reliable distributed coordination. We have investigated whether a useful group key management service can be

built using ZooKeeper. PVeStA statistical model checking showed that such a ZooKeeper-based service handles faults better than a traditional centralized group key management service, and that it scales to a large number of clients while maintaining low latencies.

To the best of our knowledge, the above-mentioned work at the Assured Cloud Computing Center represents the first published papers on the use of formal methods to model and analyze such a wide swathe of industrial cloud storage systems.

Section 5 Conclusions

We have summarized our experience using rewriting logic and Maude, with its extensions and tools, as a suitable framework for formally specifying and analyzing both the correctness and the performance of cloud storage systems. Rewriting logic is a simple and intuitive yet expressive formalism for specifying distributed systems in an object-oriented way. The Maude tool supports both simulation for rapid prototyping and automatic ``push-button" model checking exploration of all possible behaviors from a given initial system configuration. Such model checking can be seen as an exhaustive search for ``corner case" bugs, or as a way to automatically execute a more comprehensive ``test suite" than is possible in standard test-driven system development.

Furthermore, PVeStA-based statistical model checking can provide assurance about quantitative properties measuring various performance and quality of service behavior of a design with a given confidence level, and Real-Time Maude supports model checking analysis of real-time distributed systems.

We have used Maude and Real-Time Maude to develop quite detailed formal models of a range of industrial cloud storage systems (Apache Cassandra, Megastore, and Zookeeper) and an academic one (RAMP), and have also designed and formalized significant extensions of these systems (a variant of Cassandra, Megastore-CGC, a key management system on top of ZooKeeper, and variations of RAMP) and have provided assurance that they satisfy desired correctness properties; we have also analyzed their performance. Furthermore, in the case of Cassandra, we compared the performance estimates provided by PVeStA analysis with the performance actually observed when running the real Cassandra code on representative workloads; they differed only by 10-15%.

Section 6 Intrusion Detection, Response, and Recovery in the Cloud (William H. Sanders, Atul Bohara, and Uttam Thakore)

Section 1 Summary of Research Project

This report presents our work on intrusion resilience of distributed systems such as clouds and enterprise networks. First, we build an actor-centric, asset-based model for cloud security threats. Using the proposed model, we identify the layers in the cloud that each threat affects to assist practitioners in targeting defenses. Next, we propose a quantitative methodology to determine optimal monitor deployment. The approach is based on admin-specified intrusion detection goals and cost constraints. Subsequently, to fuse and analyze the diverse information recorded by the monitors to achieve intrusion resilience goals, we propose the following techniques:

1. We enable highly-relevant prioritization of monitor data for analysis that aids administrators with the real-time incident response. Our approach uses statistical causality analysis techniques on monitor data to identify information that would promote earlier detection of incidents, without relying on administrator labeling.
2. To improve the detection of flooding-based network attacks on enterprise systems, we propose an unsupervised cluster analysis and prioritization approach using features extracted from host-level authentication logs and network-level firewall logs. We compare the feature distributions of different clusters to identify anomalous behavior. We then propose a cluster difference metric that we use to prioritize the anomalous clusters based on their likely maliciousness.
3. We formally define a framework for distributed fusion of host and network-level events. We use the framework to detect network-wide lateral movement behavior with low overhead in performance.
4. We correlate the lateral movement detection with command and control indicators to identify infected hosts. The approach uses an ensemble of anomaly detectors to have an accurate detection even when attacker deviates from assumed threat model.

Section 2 Introduction

Cloud providers and enterprise systems today face a broad range of attacks, both from outside agents and potentially from their consumers or users. In particular, sophisticated and large-scale attacks, such as those on Target Corp. in 2013 and Anthem Inc. in 2015, result in the theft of the personal information of tens of millions and tremendous losses for the companies breached. Such attacks are called advanced persistent threats (APTs) for the long duration of the attack and the stealth and skill used in evading detection and compromising the system.

In large enterprise and cloud systems, monitors and sensors can generate upwards of tens to hundreds of terabytes of heterogeneous data on system behavior per day. The burden of deciding

what information to collect and how to analyze it falls on system administrators and security analysts, who must use their domain knowledge to manually adjust system monitoring as attacks are missed and monitor data goes unused. These administrators and analysts must also perform real-time root cause analysis and after-the-fact digital forensic analysis when an incident has taken place to uncover the underlying vulnerabilities in the system, and the sheer diversity and volume of data can make efficient investigation difficult.

Existing security mechanisms are disjointed and noisy, and do not provide a security administrator the ability to observe the actual attacks occurring on their systems easily. Furthermore, existing mechanisms assume that monitor data can be completely trusted, which is a false assumption when the monitoring infrastructure resides on the system being attacked. This project aims to allow cloud providers and enterprise security administrators to quickly and effectively decide what information to collect in the system; how to fuse the heterogeneous data to discover behaviors otherwise difficult to detect, such as lateral movement during an APT; and how to best present the information to administrators to help them detect intrusions and respond to them in real time. Ultimately, we aim to help administrators provide stronger security assurances to their stakeholders.

Our approach is to use both model-driven and data-driven techniques to make monitoring and fusion decisions. We use the system model, motivated by system topology, vulnerabilities, attack surface, and response mechanisms, to drive the placement of monitors and fusion agents. We further use many semi-supervised and unsupervised learning techniques to analyze monitor data and fuse it in new and meaningful ways to discover new information about the system and improve our likelihood of intrusion detection. Ultimately, we provide administrators with improved context and visibility into their systems in real-time by analyzing, correlating, and prioritizing data collected from many different instrumented components of the system.

Section 3 Methods, Assumptions, and Procedures

In this section, we describe the proposed methodology for security monitoring, data fusion, and intrusion detection.

Monitors and Monitor Deployment:

In this work, we address the problem of improving security monitoring and monitor data analysis techniques for enterprise and cloud systems. Specifically, we help system administrators determine what data to collect within their system and how to target analysis efforts to maximize their effectiveness in detecting and responding to intrusions.

This work consists of the following components: (1) quantifying the utility of security monitor data in detecting intrusions, (2) devising a methodology to determine an optimal placement of

monitors based on intrusion detection requirements, and (3) devising a mechanism by which data collected from security monitoring can be correlated and analyzed dynamically based on ongoing observations and in tandem with input from system administrators.

To start, we create an actor-centric, asset-based cloud threat model that enables practitioners to reason about monitor deployment regarding the security of their cloud assets [5]. We define an actor model that consolidates several roles in the literature to three roles motivated by security. We also define an architectural model that identifies the assets that can be owned by each actor and use it to create an asset-based cloud threat model. Our threat model promotes reasoning about cloud monitor deployment, and motivates our subsequent work.

One of the assumptions that drives our work is that security monitor data can be unreliable, which is a known issue in large enterprise and cloud systems. The first move a stealthy attacker will make is to disable monitoring or hide its tracks to hinder the ability of security analysts to observe and detect its malicious activity. This understanding motivates our research in two ways. First, we consider the effect of monitor compromise in the development of our metrics and analysis techniques. Second, we strive for increased redundancy in data collection from many levels of the system to increase the ability to detect intrusions when some monitors are unavailable or are providing erroneous data.

Furthermore, we observe that different monitors produce heterogeneous information. As a result, we take heterogeneous information into account when representing the information produced by monitors and consumed by intrusion detection systems and forensic analysts.

Motivated by our threat model, we develop a methodology to both evaluate enterprise and cloud monitor deployments quantitatively in terms of security goals and to deploy monitors optimally based on cost constraints [3]. First, we define a model that describes the system assets, deployable monitors, and the relationship between generated data and intrusions. Then, we define a set of metrics that quantify the utility and richness of monitor data with respect to intrusion detection and the cost associated with deployment. Finally, we formulate a nonlinear, 0-1 optimization problem using our model and metrics to determine the cost-optimal, maximum-utility placement of monitors. In our optimization problem, we attempt to find a monitoring deployment that meets security administrator cost and utility constraints and maximizes the utility of the administrator. We define a set of weights and constraints on the monitoring metrics that an administrator can use to specify intrusion detection requirements, and define the monitor utility as a weighted sum of our monitoring metrics. To solve the optimization problem, we develop a branch and bound algorithm, then create a greedy approximation that would scale to large systems.

Finally, we start to move towards a data-driven approach to quantifying the utility of monitors. We are working on a method to provide dynamic monitoring recommendations as incidents take place and aid administrators in incident investigation during system operation by analyzing diverse monitor data using statistical methods. In this work, we use statistical correlation tests, which identify strong correlations between features that may be attributed to causal relationships between the features, to identify time-lagged correlations between cloud logging and monitoring data sources when specific incidents take place. Using correlation techniques best suited to the data sources being analyzed, we construct a temporal chain of strongly correlated relationships between logging and monitoring data sources for a given incident extending backwards from an incident's occurrence, which we use to prioritize the importance of the data sources for data collection and analysis by security and operations administrators.

Fusion and Analysis:

In this work, we build methods to combine and analyze diverse monitor data to improve intrusion detection for enterprise and cloud systems.

First, we present an approach to identifying anomalous behavior in the unlabeled system log and network log data using unsupervised machine learning [4]. We first establish a threat model and extract meaningful features from the log data that aid in attack detection. We then describe a method to combine the data using features we have defined that allows us to perform anomaly detection over the joined network and host log data. Next, we use the k-means and DBSCAN clustering algorithms to categorize the data into different usage profiles, and we compare the feature distributions of different clusters to identify anomalous behavior. We then propose a cluster difference metric that we use to prioritize the anomalous clusters based on their likely maliciousness. Finally, we manually analyze the clusters to correlate them with known attacks and evaluate our approach.

Next, we develop a flexible framework for distributed data fusion aimed at addressing intrusion resilience [2]. The framework defines different components of data fusion: data transformation, dissemination, and abstraction. We use the framework to define a method that uses agents in the system to detect lateral movement. Our method merges host-level communication causation events to create host communication graphs. The merge algorithm exploits the semantics of the causation relation to avoid requiring time ordering on the host-level events. Then, to avoid having a centralized collection agent, we cluster the agents into a hierarchy in which each cluster leader maintains local host communication graphs and sends abstracted updates to the global collection agent.

Finally, we propose an unsupervised multi-detector approach to detect lateral movement-based attacks in enterprise systems [1]. We extract useful features using NetFlow, C&C, and specialized lateral movement monitors. We then propose a graph-based model to combine the

diverse features to evaluate current security state of the system. We show that by using the proposed features with an ensemble of PCA, k-means clustering, and extreme value analysis enables the identification of compromised hosts in an unsupervised manner.

Section 4 Results and Discussion

To demonstrate the utility of our asset-based, actor-centric cloud threat model, we applied our model to analyze security-driven monitor deployment in an artificial but realistic cloud architecture based upon Netflix's use of Amazon Web Services [5]. This work motivated our quantitative methodology for security monitor deployment.

We illustrated how our monitor deployment methodology could be used in practice to determine optimal monitor deployments by constructing a case study using a set of common attacks on Web servers in an enterprise Web service use case [3]. We built the case study model using common Web service softwares – specifically, we set up a distributed LAMP stack running on top of Metasploitable Kali Linux instances, and ran the Mutillidae web service on top of it. To obtain the data model, we attacked the service and collected network and application log, then mined them for indicators and their mappings to monitors. We then demonstrated the scalability of our methodology and optimization problem heuristic solution algorithm by simulating systems with hundreds of monitors and attacks and showing that our approach can compute optimal monitor deployments for such systems within minutes.

We have been evaluating our cloud monitor prioritization technique on IBM Watson Health Cloud testing data sources, which contain primarily reliability and performance incidents. We find that our approach can prioritize data sources for analysis without the need for administrator input or labeling, but our experiments are ongoing.

We evaluated the efficacy of our approach to cluster-and-combine diverse logs to detect enterprise system intrusions [4]. We used a publicly available enterprise network dataset. The proposed approach detected all attacks present in the data set except those that require additional information to detect. Using the joined network-level and host-level data, our approach could detect attacks that were not detectable with either data source alone.

We used trace-based evaluation to study the performance trade-offs of the proposed lateral movement detection framework [2]. We simulated lateral movement over a network topology and ran our fusion algorithms using the simulation traces. We evaluated the scalability of the hierarchical fusion by implementing different host-clustering techniques and computing fairness and locality metrics. Our results show that clustering methods that utilize network topology achieve a good balance between performance and quality of the state. We also implemented a

prototype of the host-level agent and found that the agent is lightweight and suitable for practical use.

We performed experiments to evaluate the accuracy of the proposed anomaly detection approach in identifying compromised hosts [1]. The experiments used internal network flow logs obtained from the Los Alamos National Lab (LANL)'s Comprehensive Multi-Source Cyber-Security Events dataset and simulated traces of lateral movement activity. Our results show that the proposed approach could accurately detect the infected hosts with a small false positive rate. The insights that we obtained during the sensitivity analysis of the proposed approach can be used by researchers to test new defense mechanisms before deploying them on real systems.

Section 5 Conclusions

In this project, we devised a quantitative monitor deployment framework that outputs a deployment plan by optimizing metrics such as coverage, cost, and confidence. The monitor deployment framework formalizes the process of choosing which monitors to deploy for protection. We also developed a technique to aid enterprise cloud security administrators in investigating performance, reliability, and security incidents during system operation by using statistical correlation tests on diverse cloud monitor data to prioritize the importance of the data sources for data collection and analysis. Furthermore, we developed methods for distributed detection of lateral movement chains using process communication graphs; the method combines process information and network information. The work on lateral movement used process communication graphs to infer network event relations previously inferred only through timing information. Finally, we developed multiple anomaly detection methods: (1) unsupervised clustering of diverse network and host logs to detect flooding-based network attacks, and (2) ensemble of multiple anomaly detectors to identify hosts that are part of malicious lateral movement.

Section 6 Bibliography

Conference and Journal Publications

- [1] Atul Bohara, Mohammad A. Nouredine, Ahmed Fawaz, William H. Sanders, “An Unsupervised Multi-Detector Approach for Identifying Malicious Lateral Movement”, [Submitted for publication].
- [2] Ahmed Fawaz, Atul Bohara, Carmen Cheh, and William H. Sanders, “Lateral Movement Detection using Distributed Data Fusion”, *35th Symposium of Reliable Distributed Systems (SRDS 2016)*, Budapest, Hungary, September 26-29, 2016.
- [3] Uttam Thakore, Gabriel A. Weaver, and William H. Sanders, “A Quantitative Methodology for Security Monitor Deployment”, *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*, Toulouse, France, June 28-July 1, 2016. **Best Paper Award.**
- [4] Atul Bohara, Uttam Thakore, and William H. Sanders, “Intrusion Detection in Enterprise Systems by Combining and Clustering Diverse Monitor Data”, *Symposium and Bootcamp on the Science of Security (HotSoS 2016)*, Pittsburgh, PA, April 20-21, 2016.
- [5] Uttam Thakore, Gabriel A. Weaver, and William H. Sanders, “An Actor-Centric, Asset-Based Monitor Deployment Model for Cloud Computing”, *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013)*, Dresden, Germany, December 9-12, 2013.

Theses

- [1] Uttam Thakore, “A Quantitative Methodology for Evaluating and Deploying Security Monitors,” MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, August 2015.

Section 7 Map-Reduce Task Assignment with Data Locality Constraint (Yi Lu)

Section 1 Summary of Research Project

Task scheduling for a system with data locality constraint is an important problem as it plays a significant role in system efficiency, which can be evaluated by throughput and job completion time. We are interested in designing a scheduling algorithm that achieves throughput optimality and fast task completion.

Section 2 Introduction

This research focuses on improving the scalability and efficiency of large data analytic systems in the software layer.

While the current data analytic systems, such as the Hadoop ecosystem, employs the horizontal scaling philosophy and makes it easy to build a more powerful system simply by adding servers, the bottleneck lies in the master node responsible for scheduling and resource management. The problem is exacerbated by the new demand of fast, on-line queries, which are key to accelerating discoveries, but makes the overhead of a centralized scheduler excessive.

A naïve solution is to segregate the cluster into disjoint sub-clusters, each with its own central scheduler, but this will require a much higher level of data replication, wasting storage and increasing file system maintenance overhead. An alternative will be an array of distributed schedulers, each with full knowledge of the cluster status. This will not change the amount of data replication, but will require a high communication overhead, as each server in the cluster needs to communicate with each of the distributed schedulers.

We propose to investigate a solution with distributed schedulers using the idea of “reverse information balancing”, with no increase in data replication and only a slight increase in communication overhead compared to a centralized scheduler. We have managed to show its superiority in performance in a homogeneous cluster. The challenge is to demonstrate its efficiency and scalability in the data analytic environment where each server is different from another due to the “data locality” constraint: It is much more efficient for a server to process a task with data stored locally than one with data stored remotely, and each server stores a different combination of data chunks due to availability concerns. Our initial experiments have shown promising results.

Section 3 Methods, Assumptions, and Procedures

We investigated some existing approaches to map task scheduling: The FIFO scheduler maximizes utilization, delay scheduling maximizes data locality, and the JSQ-MaxWeight algorithm pre-assigns remote tasks. However, except for the throughput optimality of JSQ-MaxWeight, very little is known about the relative performance of the three approaches. Part of our work is to study the performance of these existing algorithms.

We proposed a novel algorithm that is throughput-optimal and achieves fast task completion. The main idea is to use a queueing structure within the scheduler to 1) load balance local tasks across servers to reduce unnecessary idling; 2) identify the overloaded servers and offer timely remote service; 3) provide estimates of a task's waiting time to help with remote task assignment: it prevents unnecessary remote task assignment in FIFO scheduler that results in longer completion time and also prevents unnecessary waiting in the delay scheduling.

In order to make the algorithm robust, we do not assume the knowledge of traffic intensity or the distribution of requested data on servers. In particular, to evaluate the proposed algorithm against existing algorithms, we considered two distinct scenarios for simulation: 1) uniform workload case where the data requested by the incoming traffic are uniformly distributed on servers; 2) skewed workload case where the data requested are skewed towards a subset of servers, which become hotspots at high load.

Section 4 Results and Discussion

As for the existing algorithms, we have following findings:

- We proved that the task-level algorithm of the FIFO scheduler is throughput optimal.
- We showed that while delay scheduling yields fast completion with a uniform load, the stability region it achieves is much smaller than full capacity region with skewed load distribution.
- For completion time, the simulation results showed that the task-level FIFO scheduler performs poorly at low and medium load, and JSQ-MaxWeight performs poorly at medium and high load.

For the proposed algorithm:

- We established the throughput optimality of the proposed algorithm.
- Simulation results for the throughput-optimal algorithms showed that the proposed algorithm outperforms both FIFO and JSQ-MaxWeight at all loads. It achieves 2-4 fold improvement over the other algorithms in completion time.

Section 5 Conclusions

We have shown that the proposed algorithm is heavy-traffic optimal, which makes it the only known heavy-traffic optimal algorithm for all load distributions.

Section 6 Recommendations

- Future work:
 - We are planning on implementing the algorithm in the Hadoop framework.
 - Our current results are based on the assumption of Poisson arrival and exponential service time distribution. We would like to study the proposed algorithm without these assumptions.
 - We are investigating the response time distribution at any load for a system with data locality

Section 8 Security and Privacy Mechanisms: An Analysis of Certifications for Federal Cloud Service Providers (Masooda Bashir and Carlo Di-Giulio)

Section 1 Summary of Research Project

To demonstrate compliance with privacy and security principles, information technology (IT) service providers often rely on security standards and certifications. However, new service models such as cloud computing have brought new threats to information assurance.

In the first section of our project, we analyze four of the most highly regarded information technology security certifications used to assess cloud security. Those are ISO/IEC 27001, SOC2, C5, and FedRAMP. We describe the evolution of those four security standards and the improvements made to them over time to cope with new threats, and focus on their adequacy and completeness by comparing them to each other.

In the second section of our research project, we focus the attention on FedRAMP. We evaluate resilience and completeness of the standard in relation to new technology such as containers. The additional threats that derive from using containers make existing security standards inadequate, and FedRAMP needs to include additional elements to cope with those new vulnerabilities.

Section 2 Introduction

The goal of the research is to help identify some of the most secure, and define possible limits in the usage of, cloud services provided to the Air Force by private outsourcers.

In the first section of our research, our focus is on privacy and security standards and certifications for cloud services. The creation of the Federal Risk Authorization Management Program and equivalence of its baseline with DISA security level (moderate baseline to level 1-2, and high to 3-4) makes of FedRAMP requirements a topic of extreme relevance for DoD agencies. Significant costs for Cloud Service Providers in the achievement of a FedRAMP authorization, and the existence of concurrent internationally recognized IT security standards has raised questions about the necessity of a new certification such as FedRAMP: how effective are current IT security measures and standards at addressing cloud security? Is FedRAMP better than ISO/IEC 27001, SOC 2, or the recently developed C5 at protecting information assurance in cloud environments, and if so, how? Is it ultimately worth it to invest in new cloud security standards like FedRAMP? To answer to these questions, we have concentrated our effort in evaluating the effectiveness of FedRAMP at a moderate and high baseline compared to ISO/IEC 27001, SOC2, and C5 in terms of addressing IT security needs in cloud computing. The results of our observations have clarified the impact of FedRAMP on the vast landscape of IT security standards. By observing the evolution of FedRAMP, ISO/IEC 27001, SOC2, and C5 in more

than a decade since their first releases, we are able to draw further conclusions about their timeliness and adaptability. Through a systematic, combined review of the four standards we offer a comprehensive picture of their completeness and adequacy. We highlight missing controls and control domains. In our conclusions, we offer evidence of the existence of weaknesses in the certification build process and suggest improvements to the effectiveness of further versions of these frameworks.

In the second section of our research, we consider the appearance of newer technology in cloud environments and the potential risks associated with it, which make of information assurance a top priority for Federal Agencies and IT Industry. We move from the results obtained in the first part of our research to answer to several more questions: what resilience do standards such as FedRAMP show when new technology is used in cloud environments? What sort of adjustments does the standard need to be considered adequate?

One of the most revolutionary innovations in cloud computing in the last years has been the adoption of containers replacing virtual machines to host tenants' applications. Containers can optimize the virtualization capabilities of the host, reducing the consumption of resources supporting the applications, and sensibly improving their response time. However, existing IT security standards, and in the specific FedRAMP, are not designed for container technology, and their lack of adaptability might cause limitations, or even worse flaws in the certification process. To answer to our questions, and determine resiliency of FedRAMP in relation to container technology, we analyze existing standards, laws, and guidelines to find flaws and gaps in the current regulatory landscape. For example, a gap-analysis of FedRAMP and NIST SP 800-53 (Security and Privacy Controls for Federal Information Systems), helps highlighting missing controls and measures useful to adequately respond to the use of containers instead of virtual machines.

Section 3 Methods, Assumptions, and Procedures

In the first section of our project, we rely on a mixed quantitative and qualitative method. The methodology can be divided in three steps.

1. We collect the controls in FedRAMP, SOC2, ISO/IEC 27001, and C5 and compared them against a third-party framework, the Cloud Control Matrix (CCM) by the Cloud Security Alliance **Error! Reference source not found.** The CCM offers the advantages of being focused on cloud security and being easily comparable with the other standards because of its structure, which is based on controls and control families like the standards.
2. We analyze the differences in the numbers of controls from the CCM that are omitted in each of the four standards. That allows us to build a quantitative comparison among the standards, highlighting their shortcomings. We compare all the available versions of the standards to obtain an historical perspective on their adequacy in addressing cloud security.

3. We narrow down the analysis to the most relevant controls, selected from those having a direct impact on the particularly critical risks the CSA identified as the “treacherous twelve”

In the second section of our project, we focus our attention to the security controls listed in NIST SP 800-53 useful to increase security of cloud systems using container technology and missing in FedRAMP. We narrow the scope of our study to specific areas where countermeasures to container vulnerabilities could be applied. We select three areas from six suggested in NIST SP 800-190, Hardware, Host OS, and Orchestrator level. In this section of the project, the methodology follows three steps.

1. We analyze the controls and enhancements in NIST SP 800-53 **Error! Reference source not found.** to identify those relevant to protect against container vulnerabilities. Using a full-text search of keywords in the “control description” and “guidance” fields of the controls in NIST SP 800-53, we reduce the number of controls in NIST 800-53 to some of the most relevant for container technology. In addition, we include all the controls referred in NIST 800-190 as relevant for container technology.
2. Of all the selected controls and enhancements, we keep only the ones we consider relevant as countermeasure in the three areas identified as the subject of our study (Hardware, Host OS, Orchestrator). We do not consider controls resulting in general security measures, applicable regardless of whether the configuration of cloud systems is based on containers or virtual machines. The evaluation is on the effectiveness of the control to be particularly impactful in case of containerization.
3. We remove the controls included in FedRAMP moderate baseline and focus on controls adopted only in the high baseline, or completely excluded from FedRAMP.

Section 4 Results and Discussion

In the first section of our study, we analyze FedRAMP, ISO/IEC 27001, SOC 2, and C5 and their response to cloud security threats. Out of 133 controls in CSA’s Cloud Control Matrix, the third-party framework used for our comparison:

- **SOC 2** - The three versions of TSPC, published in 2009, 2014, and 2016, and specifying the criteria for SOC 2 assessments show 43, 47, and 39 omissions, respectively;
- **FedRAMP** rev. 3, released in 2012, shows 45 omissions, while the 2015 release 4 of FedRAMP shows a significant improvement with 29 missing controls;
- **ISO/IEC 27001** satisfies all but 43 and 3 controls in its 2005 and 2013 releases, respectively;
- **C5**, although building on the ISO certification and TSPC to define its own set of criteria, shows as many as 30 omitted controls across multiple control domains.

Interestingly, two control domains are completely or substantially omitted in most of the frameworks we analyzed. The first domain is Mobile Security (MOS). The second domain is

Interoperability and Portability (IPY). Only ISO/IEC 27001:2013 addresses all the controls in those domains. When we consider the relevance of the omitted controls according to their impact on at least one of the treacherous twelve threats identified by the Cloud Security Alliance, on a total of 83 controls relevant for the Treacherous Twelve:

- **SOC 2** – TSPC show a fluctuation suggesting that the older version (from 2009) offers better protection than the newer ones. The TSPC from 2009, 2014, and 2016 omit 10, 16, and 12 controls, respectively;
- **FedRAMP** rev 3 omits 11 controls, while rev. 4, goes to only 5 omissions;
 - we acknowledge that of the five controls missing in FedRAMP only one finds mitigation in other measures prescribed in FedRAMP itself or Federal acts. In particular, FedRAMP oversees mobile security, missing controls on Bring Your Own Device (BYOD) policies and automatic lock screen measures.
- **ISO/IEC 27001** - ISO/IEC 27001:2013 goes from 3 to 2 omitted controls. ISO/IEC 27001:2005 still omit 10 controls;
- **C5** drops the number of omitted controls to 7.

If we focus on the controls relevant for the Treacherous Twelve, the absence of controls in the MOS and IPY domains largely accounts for the drop in numbers of missing controls. The same absence justifies the limited variation in ISO/IEC 27001:2013 that covers both domains. Of 83 controls impacted by the Treacherous Twelve threats identified by CSA, 63 are addressed in all the standards, thus we affirm that there is not a radical difference in their substance. All four of the frameworks are aimed at providing information assurance in IT systems through a similar set of baseline controls. The four standards are complementary, as they overlap significantly and demonstrate to be interchangeable when it comes to cloud assurance. On the other hand, a total of nineteen controls are missing from at least one standard; of those, only four are missing from more than one standard. That highlights some relevant differences in the standards' approaches to IT security.

In the second section of our project, we focus our attention on FedRAMP and its resiliency to new technology such as containers, and new vulnerabilities. Of 922 controls in NIST 800-53, which include control enhancements and withdrawn controls, we do a first selection based on keywords.

- We identify 222 controls through twenty-three keywords or key-phrase (e.g. "Operating System"), and we add 16 more controls specified in the Draft NIST SP 800-190.
- We select 44 controls as being relevant to the three areas included in our study: Hardware, Host OS, and Orchestrator countermeasures;
- We isolate 30 relevant controls not included in FedRAMP moderate baseline.

Although some vulnerabilities are shared between VMs and containers, and some controls could be used to add security to VMs as well, the more insecure nature of containers makes necessary for standardization bodies and auditors to consider additional security measures, and perform more stringent controls compared to VMs. Those controls only included in FedRAMP high baseline could be included at a lower tier.

Section 5 Conclusions

CSPs might benefit from our completeness and adequacy assessment of each standard to determine which framework is the most appropriate to evaluate their cloud security.

We have determined that the examined standards are not completely interchangeable, but rather complementary. On the one hand, that finding justifies the existence of multiple standards, as they can be used in combination to guarantee cloud assurance. Since each of them proposes a slightly different approach to assessment and auditing, and focuses on different aspects of IT security, compliance with more than one framework allows a CSP to perform a more comprehensive and nuanced audit of its systems. On the other hand, little effort would be required to improve the standards, adding missing measures to prevent more vulnerabilities or threats.

In regard to FedRAMP, we have shown how it could be improved with greater attention to information management policies and procedures. Additional attention must be given to mobile security, which is a flaw not only in FedRAMP, but in three of the four standards. Clarity in the definition of bring-your-own-device policies is the main issue in mobile security, since the absence of well-defined rules could generate (or amplify the magnitude of) insider threats.

At the same time, FedRAMP results insufficient to provide full protection against new vulnerabilities deriving from the use of containers. The nature of containers, less secure than virtual machines, justifies the addition to the moderate security level of controls currently included only in the high baseline; nonetheless, controls not currently included in FedRAMP at all must be included to ensure better protection of information and applications residing in cloud systems.

Section 6 Bibliography

[1] Carlo Di-Giulio, Read Sprabery, Charles Kamhoua, Kevin Kwiat, Roy Campbell, and Masooda Bashir, “Cloud Standards in Comparison Are New Security Frameworks Improving Cloud Security?”, *10th IEEE International Conference on Cloud Computing (Cloud 2017)*, Honolulu, HI, June 25-June 30, 2017, to appear.

[2] Carlo Di-Giulio, Read Sprabery, Charles Kamhoua, Kevin Kwiat, Roy Campbell, and Masooda Bashir, “IT Security and Privacy Standards in Comparison: Improving FedRAMP Authorization for Cloud Service Providers”, *International Workshop on*

Assured Cloud Computing and QoS Aware Big Data (WACC 2017), Madrid, Spain, May 14, 2017, to appear.

[3] Carlo Di-Giulio, Read Sprabery, Charles Kamhoua, Kevin Kwiat, Roy Campbell, and Masooda Bashir, “Cloud Security Certifications: A Comparison to Improve Cloud Service Provider Security”, *International Conference on Internet of Things, Data and Cloud Computing (ICC 2017)*, Churchill College, Cambridge, UK, March 22-23, 2017.

[4] Bashir, M. and Di Giulio, C. “Certifications Past and Future. Future model for assigning certifications that incorporate lessons learned from past practices”. In *Assured Cloud Computing*. Edited by Roy H. Campbell. IEEE Press. Forthcoming 2017.

[5] Masooda Bashir, Jay P. Kesan, Carol M. Hayes, and Robert Zielinski; “Privacy in the Cloud: Going Beyond the Contractarian Paradigm”, *2011 Workshop on Governance of Technology, Information, and Policies (GTIP 2011)*, Orlando, FL, December 6, 2011.

[6] Mullens, C., Kesan, J., Hoff, K. & Bashir, M. (2015, 25-27 September). Shaping privacy law and policy by examining the intersection of knowledge and opinions. Paper to be presented at the Research Conference on Communication, Information and Internet Policy (TPRC 43), Arlington, Virginia.

http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2588077

[7] Mullens, C. Kesan, J., Hoff, K. & Bashir, M. (2014, 12-14 Sept.). Knowledge, behavior, and opinions regarding online privacy. *Proceedings of the 42nd Research Conference on Communication, Information and Internet Policy (TPRC 2014)*, George Mason University, Arlington,

VA. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2418830

[8] Hoff, K. & Bashir, M. (2014) Trust in automation: integrating empirical evidence on factors that influence trust. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 57 (3), 407-434. DOI:[10.1177/0018720814547570](https://doi.org/10.1177/0018720814547570)

Section 9 Security Data Analysis and Design of Software Architecture for Attack Containment (Ravi Iyer and Phuong Cao)

Section 1 Summary of Research Project

Our research addresses the problem of detecting real, multi-stage attacks targeting enterprise networks at an early stage to demonstrate significant reduction to system misuse and data leaks caused by attackers. Such attacks are difficult to detect by traditional Intrusion Detection Systems (IDS) because at an early stage, only partial information of the attack is available for detection. We propose a new probabilistic framework that detect such attack stage-by-stage using Factor Graphs. The key idea in our approach is to learn and incorporate dependencies among observed security events and hidden attack stages, to infer the most probable attack stage at runtime. Factor Graphs works because current multi-stage attacks share significant characteristics of their stages with past attacks, despite attackers employ novel exploitation techniques. Our framework has been integrated to Bro, a widely-used IDS, such that we provide *the first implementation of an open-source IDS that can detect multi-stage attacks*. Our results on real attacks show that majority of multi-stage attacks can be identified and stopped before system misuse.

Section 2 Introduction

Motivation. As the scale and complexity of enterprise networked systems increase, so as the number of security vulnerabilities. Traditional attacks exploit well-known vulnerabilities, such as SQL injections or buffer overflows, and often involve a single attack stage that executes a malicious command. Detection of such attacks are straight-forward using signature of malicious commands. However, a signature only works for a specific, known malicious command and often does not generalize to novel malicious commands. Motivated by financial and political gains, modern attacks are becoming increasingly sophisticated and more difficult to detect. Such multi-stage attacks gain persistent access to the target system using root-kit, use covert communication channels (C&C) to stay under the radar of security scanning tools, and extract sensitive data while the attacker maintain C&C to the target system. As a result, multi-stage attacks can result in a system being compromised for a long time, e.g., in average 205 days before the intruder is discovered, cf. FireEye Advanced Threat Report.

A real example. Figure 16 shows a multi-stage attack observed in the wild in Jan 2017. This attack started with an illegitimate login to a user account via stolen passwords and ssh keys. While in the system, the attacker logged into a vulnerable node, obtained the root permission using a kernel local privileged escalation exploit targeting Redhat Enterprise Linux (CVE-2016-5195), and deployed a rootkit (named Venom) in a memory-mounted directory `/dev/shm`. Per analysis of the VENOM rootkit from CERN, the rootkit has been active in the wild since Jan 2017. The rootkit had two components that aimed at maintaining unauthorized and persistent access to the compromised system:

- i) A user-land binary that is an encrypted backdoor that receives remote commands to launch attack payloads, and
- ii) A Linux Loadable Kernel Module (LKM) that provides a stealth port knocking service for the user-land backdoor.

Port knocking is a stealthy technique to open a port only upon receiving a specific sequence of “knocking” packets. For example, the port 9090 on a compromised machine opens only after receiving a “knocking” sequence of three TCP packets such that: “TCP src_port + seq_number = 1221”. As a result, the compromised node can stay under the radar of regular network security scanning tools.

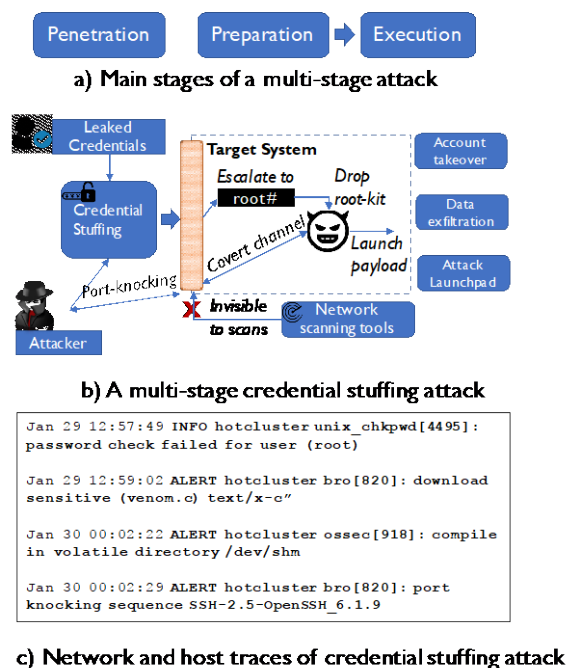


Figure 16. Attack stages and traces of a multi-stage attack using the VENOM rootkit

Problem formulation and goals. In this paper, we consider attacks that follow a multi-stage model:

- (i) a *penetration stage*, in which attackers access the victim's network, by *scanning* for valid credentials using a dictionary of leaked password or using stolen credentials, to establish an *initial compromise*;
- (ii) a *preparation stage*, in which attackers gather system information and *identify vulnerabilities* (e.g., by querying system configuration or running applications), *exploit* and *escalate* privilege to develop an attack strategy and/or tools to deploy the attack; and
- (iii) an *execution stage*, in which attackers deliver the malicious attack payload (e.g., construct and download specialized malware), *maintain* persistent presence, *clear logs*, *monitor* (using command and control), and *deliver* of attack payloads. While the model is generic, in this work we focus on a family of multi-stage successful attacks occurring at the National Center for Supercomputing Applications (NCSA, <http://www.ncsa.illinois.edu/>) over a period of nine years. Our goal is to use security events (corresponding to both alerts and other user activities, e.g., login, downloading a file, or scanning a host) observed at network and host level to infer what stage an attack is in.

Previous work and research gap. Even though critical infrastructures are equipped with multiple monitoring solutions, security violations nevertheless happen. The main causes of security violations are: i) compromised accounts allow attackers to masquerade as legitimate users, ii) session hijacking allows attackers to gain unauthorized access using a valid session key, and iii) use of remotely exploitable vulnerabilities.

Related work on multi-stage attacks have mainly focused on analysis and monitoring. *First*, analysis of vulnerabilities such as Heartbleed or VENOM rootkit provided an understanding of vulnerability used in one attack stage. However, to detect multi-stage attacks, we need to put such vulnerabilities in the context of multi-stage attacks that requires: i) not only a vulnerable library but also its complicated dependencies and ii) corresponding network and host monitoring infrastructure (e.g., network flows, system logs, or authentication logs). It presents an engineering challenge to reconstruct such vulnerable environment depending on nature of each attack. *Second*, individual network and host monitors can detect potential malicious actions in one attack stage; however, they do not detect multi-stage attacks which involve a chain of malicious activities. An individual alert in an alert chain by itself can be a false positive. For example, a host monitor may issue alert “sshd: failed password for root”, which is often a brute-force attempt and can only indicate an attempt for an initial compromise. Without correlating alerts from host and network logs, system administrators often neglect individual alerts and cannot provide a complete picture of an ongoing, multi-stage attack **Error! Reference source not found.** *Finally*, while very few work has addressed the issue of detecting multi-stage attacks, past work have focused on a specific type of attack, e.g., multi-stage denial of service attack or fishing campaigns. Techniques used for multi-stage attack detection involve manually defining attack signatures which introduced subjectivity and required extensive manual efforts.

Such techniques, however, have not been integrated to open-source IDS for immediate testing with current attacks in the wild.

Challenges. Detecting such multi-stage attacks in their early stages (penetration and preparation) presents several challenges, because attackers use valid credentials and leave no easily discernible trace, as they infiltrate a target system in the guise of legitimate users (albeit with different behavioral patterns). Thus, only partial knowledge of attacks is available at their early stages, because the attacker's activities in the system remain to be seen. A login from a remote location, for example, may simply mean that a legitimate user is connecting from outside the regular infrastructure, not necessarily that an illegitimate user is logging-in using stolen credentials. To address threats of this nature, we need to have a mechanism to estimate the attack stages based on past and current events.

Section 3 Methods, Assumptions, and Procedures

We propose a probabilistic framework that address such challenges (Figure 9.2)

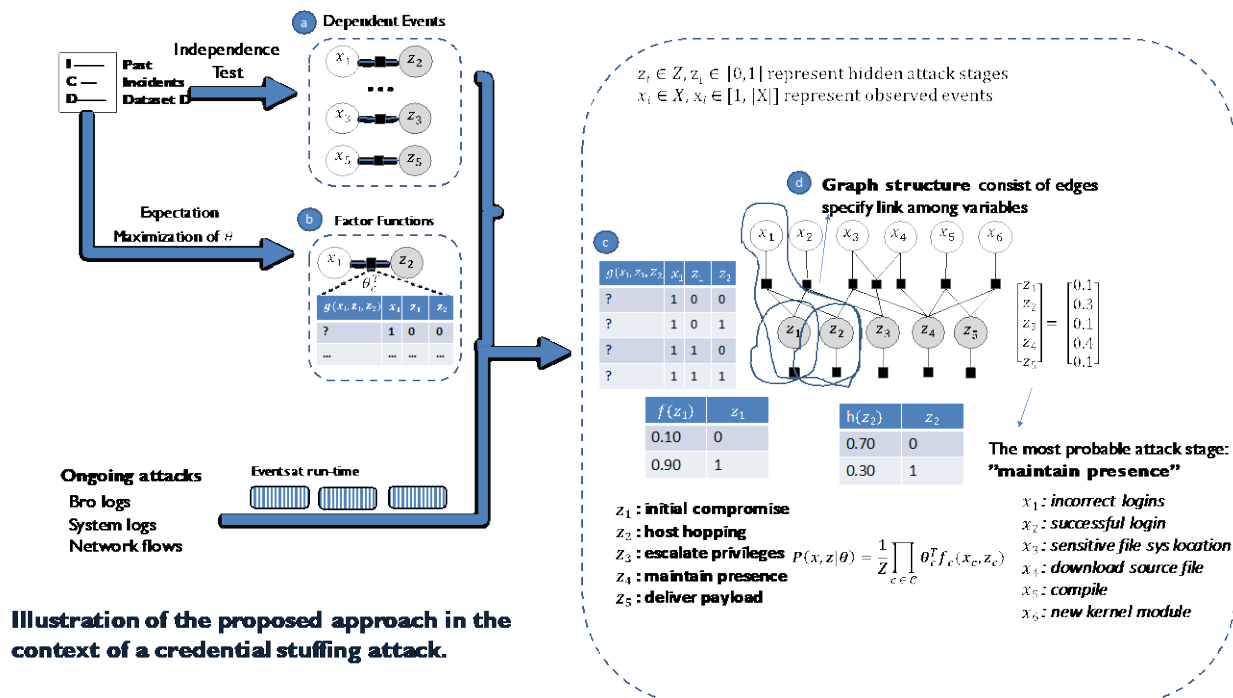


Figure 17. Workflow of factor graph framework to detect attacks from security logs

Overview. We propose a new probabilistic framework that detects an ongoing attack stage-by-stage as the attack progresses. The key idea is to infer an attack stage in a context of observed events. We represent progression of ongoing multi-stage attacks using Factor Graphs (FG), a probabilistic framework that can learn dependencies among the security events and attack stages in past attacks the form of factor functions. Legitimate user and attacker activities are represented by observed variables in a Factor Graph. Such activities are derived from security events collected at runtime by monitoring tools, such as the Bro IDS, network flows, and system

logs. Factor functions in a Factor Graph connect observed variables and hidden variables (representing a finite number of attack stages). At run-time, the most probable attack stage is inferred from the Factor Graph using belief propagation or Markov Chain Monte Carlo methods.

Our solution. The key idea behind our approach is to learn dependencies among the observed events and hidden attack stages (offline), represent such dependencies to model an ongoing attack stage-by-stage and predict the most probable stage (at runtime). Our framework takes raw host and network logs as an input, and output value of a random variable indicating the most probable attack stage.

Input data in our domain of multi-stage attack detection is illustrated in Table 2 and consists of followings:

- Raw logs of user activities at the host-level, e.g., system log, authentication log, and network-level, e.g., network flows generated by security monitoring tools.
- Ground truth data indicating an attack associated with a set of raw logs.

This input data, however, is not immediately usable by any detection model. While most IDS implement detection policies based on pattern matching in raw logs, mismatching patterns can happen and coordinating among detection policies are difficult. The main reason is that raw logs may change overtime and they are highly dependent on the underlying system. For example, a raw log for a failed password in Linux *syslog* is different from Windows Events Viewer logs, but both can be transformed into a common event `ALERT_FAILED_PASSWORD` that conveys the meaning of the underlying raw log. Thus, we seek to abstract raw logs to a set of security events.

Abstraction of security events allows our framework to work with a high level semantic of events instead of raw logs. This is a must have requirement as security monitors in enterprise networks produce a diverse set of raw logs coming from multiple operating systems. First, for each raw log line, a regular expression script automatically extract a security event e_i^t observed at time t . Thus, we can define a finite set of events $\mathcal{E} = \{e^1, \dots, e^n\}$ that defines possible security violations in a target system and use such events to construct factor functions

Accountability of security events is an important preprocessing step. Given a security event, we must determine which entity, i.e., a process, a user, or an IP address generated such events. For example, in a system log a security event can be bound to a process and a user id in a session. In our approach, we group security events by each user such that each user has a timeline of associated security events.

Ground truths for learning involve security experts to label each event with a corresponding attack stage $s_i \in S$. As a result, the data preprocessing step outputs a timeline $\mathcal{T} = [\tau_1, \dots, \tau_n]$, each entry τ_i corresponds to a raw log line and consists of an event e_i and an attack stage s_i . While most IDS only indicate an attack using alerts that prone to false positives, our approach is the first that provide stage-by-stage ground truth for multi-stage attacks and used that for supervised learning and prediction of such attacks.

Learning dependencies is the key in our approach in which prior probability of an attack stage and dependencies between observed alerts and attack stages are learned from past data using maximum likelihood estimation. In learning prior of attack stages, we collected data on frequency of each attack stage to construct the prior probabilities. In learning dependencies, we assess strength of the statistical relationship among observed events and hidden attack stages using independence test. Finally, parameters such as weights and return values of factor functions are learned using expectation minimization. The results of learning dependencies is a set of factor functions \mathcal{F} that are used to construct Factor Graphs.

Construction of Factor Graphs is done at runtime. As an event e_i is observed, a corresponding factor function $f_c(X_c)$ is chosen from the set of factor functions \mathcal{F} to construct a Factor Graph $FG = \{E, S, F\}$.

Section 4 Results and Discussion

Following are highlights of our results

Theoretical results.

- We proposed a novel probabilistic framework based on Factor Graphs for representation, learning, and inference on progression of multi-stage attacks. To the best of our knowledge, we are the first to apply factor graphs to security domain.

Real-world deployment results.

- We provided an immediate integration with the Bro IDS, an open source IDS, for detection of multi-stage attacks. Our multi-stage attack detector framework has been ingesting real network alerts generated by Bro in the National Center for Supercomputing Applications (NCSA) network.
- Our multi-stage attack detector framework can infer attack stages in real-time. An example confusion matrix of our inference result is given in Figure 9.3. The x-axis and y-axis shows the attack stages that we inferred. A number in a cell (x,y) represented the number prediction

Raw logs	User	Event (x)	Attack Stage (z)
unix_chkpwd[4495] password check failed for user (root)	alice	ALERT_FAILED_PASSWORD	scan
bro[820]download sensitive(venom.c) text/x-c"	alice	ALERT_SENSITIVE_HTTP_URI	gather
ossec[918] compile in volatile directory /dev/shm	alice	ALERT_COMMAND_AND_ANOMALY	persist
bro[820] port knocking sequence SSH-2.5-OpenSSH_6.1.9	alice	ALERT_PORT_KNOCKING_SEQUENCE	control

Table 5. Conversion from raw logs to events and labeled attack stage

made
by our

approach. For example, cell (x=D, y=D) has a value 9 means that there are 9 attacks in *delivery* stage that have been correctly predicted by our approach; cell (x=B, y=G) has a value 5 means that there are 5 attacks in the *gather information* stage that have been mis-predicted as in benign stage by our approach.

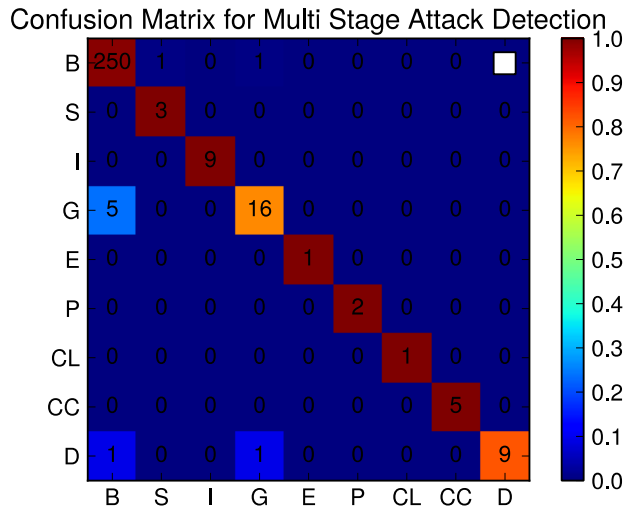


Figure 18. B-Benign, S-Scan, I-Initial Compromise, G-Gather information, E-Escalate Privilege, P-Persist, CL-Clear Logs, CC-Command & Control, D-Deliver payloads

Section 5 Conclusions

Our results on real attacks collected in the wild and security incidents at NCSA show that majority of multi-stage attacks (74%) can be identified and stopped before system misuse. While an early detection based on the most probable attack stage may carry some false detection, Factor Graphs works because current multi-stage attacks share significant characteristics of their stages with past attacks, despite attackers employ novel exploitation techniques. Even if Factor Graphs cannot resist all attacks, it allows creation of factor functions and learning from new attacks. Hence to continue to build its attack knowledge database and stop the attacks before misuse occurs.

Section 10 Test-bed for Experimental Evaluation: Design and Prototype of Techniques for Providing Cloud Error and Attack Resiliency (PI: Zbigniew Kalbarczyk (PI); graduate students: Cuong Pham, Zak Estrada, Lavin Devnani)

Section 1 Summary of Research Project

In this research, we developed a monitoring framework that addresses reliability and security in cloud computing infrastructures. We identify the commonalities between reliability and security to guide the design of HyperTap, a hypervisor-level framework that efficiently supports both types of monitoring in virtualization environments used to deploy the cloud systems. In HyperTap, the logging of system events and states is common across monitors and constitutes the core of the framework. The audit phase of each monitor is implemented and operated independently. In addition, HyperTap relies on hardware invariants to provide a strongly isolated root of trust. HyperTap uses active monitoring, which can be adapted to enforce a wide spectrum of reliability and security policies. We validate HyperTap by introducing three example monitors: Guest OS Hang Detection (GOSHD), Hidden RootKit Detection (HRKD), and Privilege Escalation Detection (PED). Our experiments with fault injection and real rootkits/exploits demonstrate that HyperTap provides robust monitoring with low performance overhead.

In order to further extend the capabilities of the HyperTap framework, we introduced and implemented the concept of *hprobes* as a basis for implementing active monitoring techniques. An *hprobe* is a mechanism used to generate an event when the target executes a particular instruction. When the target's execution reaches the *hprobe*, control is transferred to the monitoring system, which can record the event and/or inspect the system's state. Once the monitor has finished processing the event, it returns control to the target system, and execution continues until the next event. Hprobes based techniques are robust against failures and attacks inside the target when the monitoring system is properly isolated from the target system. We demonstrated usefulness of hprobes by implementing sample detectors: an emergency detector for security vulnerability, a process hang (or infinite-loop) detector, and detector of unauthorized privilege escalation. We tested our detectors on real applications and demonstrated that those detectors achieve an acceptable level of performance overhead with a high degree of flexibility.

Exploration of all these ideas paves the path to make *Reliability and Security as a Service* an actual offering from cloud providers.

Section 2 Introduction

Building resilient (i.e., reliable and secure) computing systems is hard due to growing system and application complexity and scale, but maintaining reliability and security is even harder. A resilient system is expected to maintain an acceptable level of service in the presence of internal and external disturbances. Achieving resiliency requires mechanisms for efficient monitoring, detection, and recovery from failures due to malicious attacks and accidental faults with minimum negative impact on the delivered service.

All those challenges must be tackled when building resilient cloud computing infrastructures. Prolific failures have kept reliability a leading concern for customers considering the cloud. Monitoring is especially important for security, since many attacks go undetected for long periods of time.

Cloud computing environments are often built on top of virtual machines (VMs) running on top of a hypervisor. A virtual machine is a complete computing system that runs on top of another system. The hypervisor is a privileged software component that manages the VMs. Typically, one can run multiple VMs on top of a single hypervisor, which is often how cloud providers distribute customers across multiple physical servers. As the low-level manager of VMs, the hypervisor has privileged access to those VMs, and this access is often supported by hardware-enforced isolation. The strong isolation between the hypervisor and VMs provides an opportunity for robust security monitoring. Because cloud environments are often built using hypervisor technology, VM monitoring can be used to protect cloud systems.

In addressing those challenges, this research develops resilient virtual machines to ensure protection against failures and attacks. We exploit virtualization to design and deploy low-cost highly efficient monitoring and recovery techniques that can transform a typical cloud environment into resilient computing infrastructure. In the following sections we highlight the major contributions of this work, including design and implementation of HyperTap framework and several detection/monitoring techniques (e.g., hypervisor hang detection, hidden rootkit process detection) prototyped on Linux OS.

Section 3 Methods, Assumptions, and Procedures

In order to achieve resiliency against malicious attacks and accidental failures we developed robust monitoring to provide situational awareness about the system and users' state and behavior in the context of cloud computing infrastructure. Monitoring systems can generally be split into two classes: those that perform *passive monitoring*, and those that perform *active monitoring*. Passive monitoring systems are polling-based systems that periodically inspect the system's state. These systems are vulnerable to transient attacks that occur between monitoring checks. Furthermore, constant polling of a system can be a source of unnecessary performance overhead. Active monitoring systems overcome these weaknesses since they are triggered only when events of interest occur. However, it is essential to ensure that an active monitoring system's event generation mechanism cannot be circumvented.

One class of active monitoring systems is that of hook-based systems (see Figure 19), in which the monitor places hooks inside the target application or OS. A hook is a mechanism used to generate an event when the target executes a particular instruction. When the target's execution reaches the hook, control is transferred to the monitoring system, which can record the event and/or inspect the system's state. Once the monitor has finished processing the event, it returns control to the target system, and execution continues until the next event.

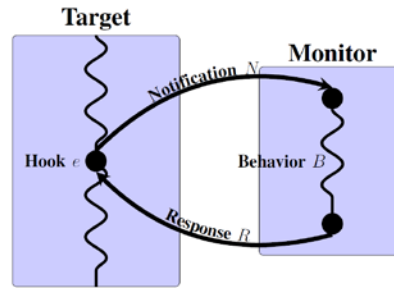


Figure 19. Hook-based monitoring. A hook is triggered by event e , and control is transferred to the monitor through notification N . The monitor processes e with a behavior B and returns control to the target with a response R .

We find dynamic hook-based systems attractive for system monitoring, as they can be easily adapted: once the hook delivery mechanism is functional, implementation of a new monitor involves adding a hook location and deciding how to process the event. In this context, dynamic refers to the ability to add and remove hooks without disrupting the control flow of the target. This is particularly important in real-world use, where monitoring needs to be configured for multiple applications and operational environments. Following those principles we designed, prototyped, and demonstrated HyperTap framework for reliability & security monitoring. Below we discuss the technologies developed.

HyperTap framework for reliability & security monitoring of virtual machines using hardware architectural invariants. We developed HyperTap, a hypervisor-level framework that efficiently simultaneously addresses both reliability and security types of monitoring in virtualization environments. In HyperTap, the logging of system events and states is common across monitors and constitutes the core of the framework. The audit phase of each monitor is implemented and operated independently. In addition, HyperTap relies on hardware invariants to provide a strongly isolated root of trust. HyperTap uses active monitoring, which can be adapted to enforce a wide spectrum of reliability and security policies. We prototype HyperTap on top of KVM hypervisor and validated the framework by introducing three example monitors: Guest OS Hang Detection (GOSHD), Hidden RootKit Detection (HRKD), and Privilege Escalation Detection (PED). Our fault injection based experiments and real rootkits/exploits demonstrate that HyperTap provides robust monitoring with low performance overhead. Evaluation results are presented in the next section.

Hypervisor Probes (hprobes) for dynamic dependability monitoring of virtual machines. We developed hprobes, a framework that allows one to dynamically monitor applications and operating systems inside a virtual Machines (VM). The hprobe framework (discussed later) does not require any changes to the guest OS, which avoids the tight coupling of monitoring with its target. Furthermore, the monitors can be customized and enabled/disabled while the VM is running. We demonstrated the usefulness of this framework by implementing sample detectors: (i) an emergency detector for a security vulnerability, (ii) an application watchdog, and (iii) an infinite-loop detector. We tested our detectors on real applications and shown that those detectors achieve an acceptable level of performance overhead with a high degree of flexibility.

Failure diagnosis for distributed systems using targeted fault injection. We developed, an approach to automate failures diagnostic in distributed systems by combining fault injection and data analytics. We use fault injection to populate a database of failures for a distributed system. When a failure is reported from the field, the database is queried to find (in the data base) execution traces similar to those of the new failure. Relying on the assumption that similar failures are caused by similar faults, we use information from the matched faults as hints to locate the actual root cause of newly reported failures. We evaluated the approach with OpenStack, a popular cloud infrastructure management system. The experimental results show that this approach can effectively determine the root causes, e.g., fault types and affected components, for 71-100% of tested failures. Furthermore, it can provide fault locations close to exact and can easily be used to find and fix actual root causes. We validate this technique by localizing real bugs that occurred OpenStack.

Section 4 Results and Discussion

In this section we provide key results from evaluation of the porotype implementation of technologies introduced in the previous section.

HyperTap framework for reliability & security monitoring of virtual machines using hardware architectural invariants.

We split the monitoring process into two phases: *logging* and *auditing*. The logging phase, when data/events are captured, constitutes the core of the framework and is common to all monitors. The auditing phase, when data/events are analyzed, is implemented and operated independently by each monitor. In order to support a broad range of auditing policies, logging needs to capture a complete view, including both actions and states of target systems. Furthermore, logging is responsible for the trustworthiness of the captured view; otherwise, auditing faces a “garbage in, garbage out” situation.

We applied the principles stated above when designing *HyperTap*, a hypervisor-level monitoring framework for Virtual Machines (VMs). In contrast to most existing VM monitoring techniques, HyperTap employs hardware architectural invariants to establish the root of trust for the logging phase. *Hardware architectural invariants* are properties defined and enforced by a hardware platform (e.g., the x86 processor architecture). Additionally, the framework supports continuous monitoring of VMs in an event-driven fashion; that enables both capturing the system state and responding rapidly to actions of interest.

We deployed and evaluated three auditors as parts of HyperTap: (i) Guest Operating System Hang Detection (GOSHD) to detect operating system hangs in a VM, (ii) Hidden Rootkit Detection (HRKD) to detect hidden malicious processes and threads, and (iii) Privilege Escalation Detection (PED) to detect privilege escalations (in Linux operating system) which allow users and applications to obtain unauthorized access to resources. Experimental evaluation of the three auditors shows that they are effective in detecting hangs (caused by injecting bugs in the guest operating system), and real-world rootkits and privilege escalation attacks while

causing less than 5% and 2% performance overhead for disk I/O and CPU intensive workloads, respectively.

Continuous Monitoring. HyperTap’s continuous monitoring takes advantage of the “trap-and-emulate” mechanism in x86 Hardware Assisted Virtualization (HAV). HAV is an extension to the x86 architecture to support running an unmodified operating system in VMs. It defines guest mode and host mode execution. In the guest mode, a processor “traps” certain privileged operations (e.g., access to processor control registers or I/O instructions) and fires VM exit events to notify the hypervisor to emulate those operations. HyperTap intercepts VM exit events, records the related VM state, and passes the collected state information to the auditor for detecting potential errors or malicious tampering with the system.

Hardware Architectural Invariants. A hardware architectural invariant, or *hardware invariant* for short, is a property defined and enforced by the hardware architecture. In most cases, these invariants must hold so that the entire software stack, e.g., the hypervisor, OS, and user applications, can operate correctly. Hardware invariants, particularly the ones defined by HAV, provide features that are desirable for VM monitoring. The behaviors enforced by HAV involve primitive building blocks of essential OS operations, such as process/application context switches, system calls, I/O accesses, and memory access events.

Implementation. The bottom left portion of Figure 20 shows a HyperTap prototype coupled with the KVM hypervisor. The same design principles can be applied to other HAV-based hypervisors. In this design, each VM can have multiple auditors of choice running at the same time. Each type of auditor can have multiple instances attached to different VMs. The core HyperTap components, including the Event Forwarder and Event Multiplexer, are responsible for delivering VM exit events to correct auditors. This design enables flexible deployment of auditors (implemented as user processes) to meet different demands of target VMs. Next we discuss auditor examples their evaluation.

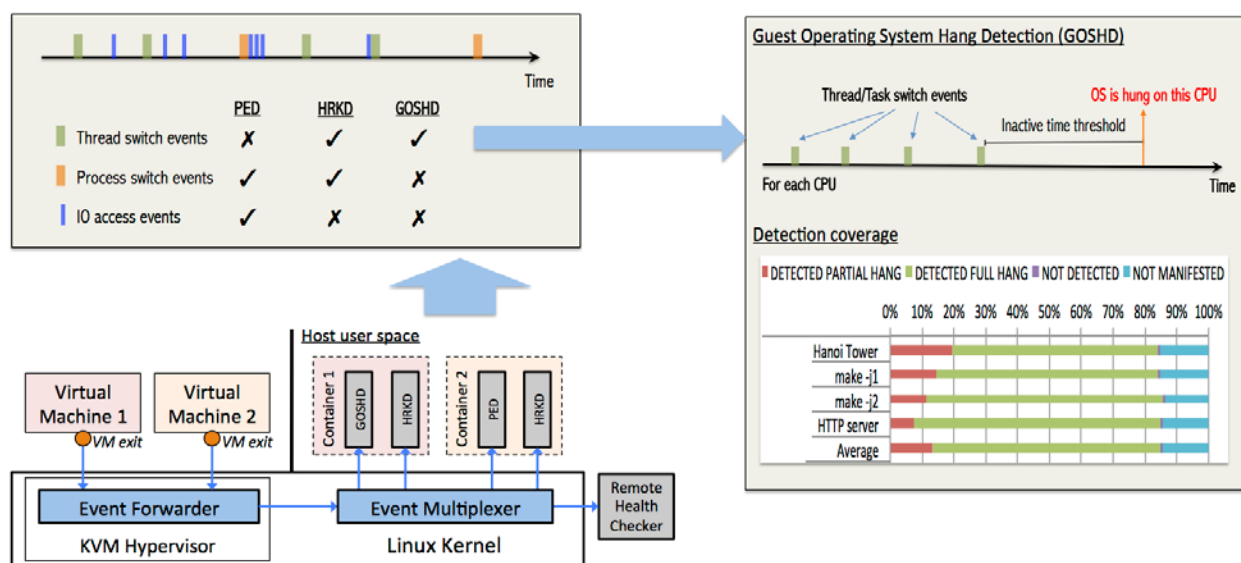


Figure 20. (Bottom left) HyperTap implementation with KVM on a Linux platform; (Top left) Event types used by three auditors; (Right) Example of GOSHD auditor; principle of operation and coverage results from fault injection experiments.

Guest OS Hang Detection (GOSHD)

Failure Model. We consider an OS as being in a hang state if it ceases to schedule tasks. In multiprocessor systems, it is possible for the OS to experience a hang on a proper subset of available CPUs. If that happens, we say that the OS is in a *partial* hang state, as opposed to a *full* hang state in which the OS is hung on all CPUs. Distinguishing between partial and full OS hangs is important, because OS hang detection approaches, such as heartbeats, are effective only against full hangs.

Detection. GOSHD tracks thread dispatches to monitor the VM's OS scheduler. If a *virtual CPU* (vCPU), meaning a CPU of a VM, does not generate any thread switch events for a predefined threshold of time, GOSHD declares the guest OS as hung on that vCPU. Because vCPUs are monitored independently of each other, GOSHD detects both partial and full hangs. The timeline in the top right portion of Figure 1 depicts the described detection mechanism.

Results. We evaluated GOSHD by injecting errors in the locking mechanisms used by Linux OS to synchronize access to shared data. The chart in the bottom right portion of Figure 2 summarizes the results. Of ~18,000 injections, about 82% manifested as hangs, and 99.8% of these hangs were detected by GOSHD. More interestingly, the experiment showed that partial OS hangs were a relatively common consequences of the injected bugs: 18% to 26% of hangs were partial hangs on preemptible and non-preemptible operating systems, respectively. That result emphasizes the importance of partial hang detection.

Hidden Rootkit Detection (HRKD)

Threat Model. Rootkits are malicious computer programs created to hide other programs from system administrators and security monitoring tools. Autonomic security scanning tools can also be bypassed simply because their inspection lists do not contain the hidden programs.

Detection. Our HRKD auditor employs context switch monitoring methods to inspect every process and thread that uses CPUs, regardless of how kernel objects are manipulated. Each time a process or a thread is scheduled to use a CPU, it is intercepted by the auditor for further inspection. That interception defeats hidden malware by putting malicious programs back on the inspection list.

Results. We tested HRKD with nine real-world rootkits in both Linux and Windows environments. In all cases, HRKD discovered the hidden applications, regardless of their hiding technique.

Privilege Escalation Detection (PED)

Threat model. A process gains a higher privilege to obtain unauthorized access to system resources. Privilege escalation is an essential step of many real-world attacks.

Detection. We employ a real-world privilege escalation detection system (Ninja) that uses passive monitoring. It is included in the mainline repository for major Linux distributions. It periodically scans the process list to determine whether a privileged (root-owned) process has a parent process that is not from an authorized user, and if it does, flags the process as privilege-escalated. We implemented two new versions of Ninja: H-Ninja and HT-Ninja. Both new versions operate at the hypervisor level. While H-Ninja uses the traditional VM monitoring method of polling and decoding VM guest memory, HT-Ninja uses architectural invariants and HyperTap to monitor VMs in an event-driven fashion.

Results. To compare the three implementations, we crafted *transient attacks*: attacks that take a very small amount of time in order to avoid being detected. We also improved transient attacks by combining them with three new attacks:

1. *Side-channel attacks*, which can determine the exact monitoring interval so that a transient attack can be timed strategically.
2. *Spamming attacks*, which increase the workload of the monitor so that the vulnerable window in which the transient attack will execute is larger.
3. *Attacks that combine a privilege escalation attack with a rootkit*, which make transient attacks persistent by hiding them from the monitor.

Using those proposed attacks, we showed that both the original Ninja and H-Ninja are highly vulnerable to transient attacks. For example, our side channel attack precisely predicted the monitoring interval of O-Ninja. Using the predicted values, we could launch transient attacks with an extremely low chance of being detected. When an attack needs more time to execute, it can utilize the spamming attack. For example, when 200 dummy processes are introduced, the detection coverage of O-Ninja is reduced to less than 2%. On the other hand, our HT-based Ninja is not vulnerable to any of those attacks, as it uses event-driven monitoring.

Hypervisor Probes (hprobes) for dynamic dependability monitoring of virtual machines

Our implementation of hprobes leverages hardware-assisted virtualization (HAV), and the prototype framework is built on the KVM hypervisor. The prototype's architecture is shown in Figure 10.3. The modifications to KVM itself make up the Event Forwarder, which is a set of

callbacks inserted into KVM's VM Exit handlers. The Event Forwarder uses Helper APIs to communicate with a separate hprobe kernel agent. The hprobe kernel agent is a loadable kernel module that is the workhorse of the framework. The kernel agent provides an interface to detectors for inserting and removing probes. This interface is accessible by kernel modules through a kernel API in the host OS (which is also the hypervisor, since KVM itself is a kernel module) or by user programs via an ioctl interface.

The execution of an hprobe-based detector is illustrated in Figure 21. A probe is added by rewriting the instruction in memory at the target address with *int3*, saving the original instruction, and adding the target address to a doubly linked list of active probes. This process happens at runtime and requires no application or guest OS restart. Although the prototype was implemented using KVM, the concept extends to any hypervisor that can trap on similar exceptions.

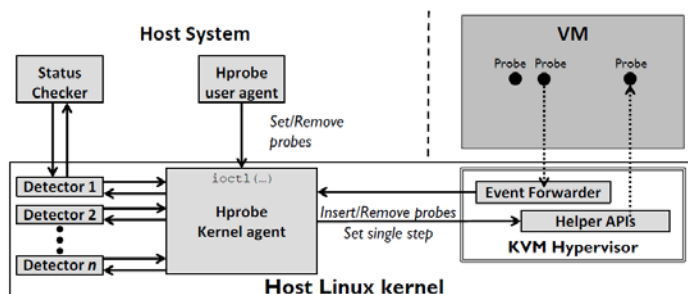


Figure 21. Hprobes integrated with the KVM hypervisor. The Event Forwarder has been added to KVM and communicates with a separate kernel agent through Helper APIs. Detectors can be implemented as kernel modules either in the host OS or in user space.

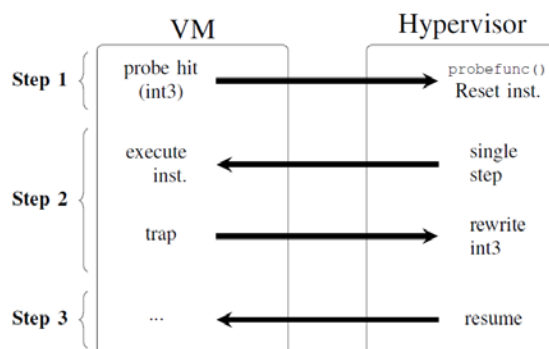


Figure 22. A probe hit in the hprobe prototype. Right-facing arrows are VM Exits, and left-facing arrows are VM Entries. When *int3* is executed, the hypervisor takes control. The hypervisor optionally executes a probe handler (*probfunc()*) and places the CPU into single-step mode. It then executes the original instruction and does a VM Entry to resume the VM. After the guest executes the original instruction, it traps back into the hypervisor, and the hypervisor will write the *int3* before allowing the VM to continue as usual.

Example Detector: *Emergency Exploit Detector*

In this section, we present *emergency exploit detector*, a sample detector built upon the hprobe prototype framework. This detector is unique to the hprobe framework and cannot be implemented on any other current VM monitoring system.

Most system operators fear zero-day vulnerabilities, as there is little that can be done about them until the vendor or maintainer of the software releases a fix. Furthermore, even after a vulnerability is made public, a patch takes time to be developed and must be put through a QA cycle. The challenge is even greater in environments with high availability concerns and stringent change control requirements; even if a patch is available, many times it is not possible to restart the system or service until a regular maintenance window. This leaves operators with a difficult decision: risk damage from restarting a system with a new patch, or risk damage from running an unpatched system.

Consider the CVE-2008-0600 vulnerability, which resulted in a local root exploit through the `vmsplice()` system call used to perform a zero-copy map of user memory into a pipe. At a high level, the CVE-2008-0600 `vmsplice()` exploit works by using an integer overflow to corrupt the kernel stack and hijack the system.

The emergency detector works by checking the arguments of a system call for a potential integer overflow. This differs in functionality from the upstream patch, which checks whether the memory region (specified by the `struct iovec` argument) is accessible to the user program. A major benefit of using an hprobe handler is that developing such a detector does not require a deep understanding of the vulnerability; the developer of the emergency detector only needs to understand that there is an integer overflow in an argument. This is far simpler than developing and maintaining a patch for a core kernel function (a system call), especially when reasoning about the risk of running a home-patched kernel (a process that would void most enterprise support agreements). Our solution uses a monitoring system that resides outside of the VM and relies on a hardware-enforced `int3` event. A would-be attacker cannot circumvent this event without having first compromised the hypervisor or modified the guest's kernel code.

Evaluation. We run microbenchmarks to estimate the latency of a single hprobe, which is the time from execution of `int3` by the VM until the VM is resumed (Steps 1–3 in Figure 10.4). We ran these microbenchmarks without a probe handler function to determine the lower bound of hprobe-based detector overhead.

Measurements were conducted on a Dell PowerEdge R720 server with dual-socket Intel Xeon E5-2660 “Sandy Bridge” 2.20 GHz CPUs (3.0 GHz turbo boost). We used a 32-bit Ubuntu 14.04 guest and measured 1000 samples. The mean latency (across samples) was found to be 2.6 μ s. In addition to the Sandy Bridge CPU, we have also included data for an older-generation 2.66GHz Xeon E5430 “Harpertown” processor (running the same kernel, KVM version, and VM image), which had a mean latency of 4.1 μ s. The distribution of latencies for these experiments is shown in Figure 5. The remainder of the benchmarks presented used the Sandy Bridge E5-2660. The hprobe prototype requires multiple VM Exits per probe hit. However, in many practical cases, the flexibility of dynamic monitoring and the reduced maintenance costs resulting from a simple

implementation outweigh that cost. The flexibility can increase performance in many practical cases by allowing one to add and remove probes throughout the VM's lifetime.

In addition to microbenchmarking individual probes, we measured the overhead of the example hprobe-based detectors presented in the previous section. All measurements were obtained using the hypercall-based timer. Our emergency exploit detector that protects against the CVE-2008-0600 `vmsplice()` vulnerability is extremely lightweight. Unless `vmsplice()` is used, the overhead of the detector is zero since the probe will not be executed. The `vmsplice()` system call is rare (at least in the open-source repositories that we searched), so zero overhead is overwhelmingly the common case. One application that does use `vmsplice()` is Checkpoint/Restart in User space (CRIU). CRIU uses `vmsplice()` to capture the state of open file descriptors referring to pipes. We used the Folding@Home molecular dynamics simulator and the pi-qmc Monte Carlo simulator as test programs. We ran these applications in a 64-bit Ubuntu 14.04 VM. At each sample, we allowed the application to warm up (load input data and start the main simulation) and then checkpointed it. The timing hypercalls were inserted into CRIU to measure how long it takes to dump the application. This was repeated 100 times for each case with and without the detector, and the results are tabulated in Table 10.1. In the table, we can see that there is a slight difference between the mean checkpoint times (roughly 3.3% for F@H and 1.7% for pi-qmc) and that the variance in the experiment with the detector active is higher for the Folding@Home case. `Sys_vmsplice()` was called 28 times when Folding@Home was being checkpointed, and 11 times for pi-qmc. We can attribute this difference to the negative cache effects of the context switch when probes are being activated.

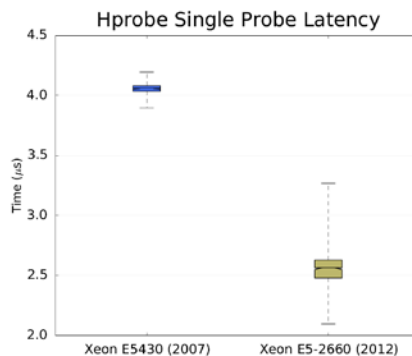


Figure 22. Single probe latency. (CPUs' release years are in parentheses.) The E5-2660's larger range can be attributed to "Turbo Boost," whereby the clock scales from 2.2 to 3.0 GHz. The shaded area is the quartile range (25th percentile to 75th percentile); whiskers are the minimum/maximum; the center is the mean; and the notches in the middle represent the 95% confidence interval of the mean.

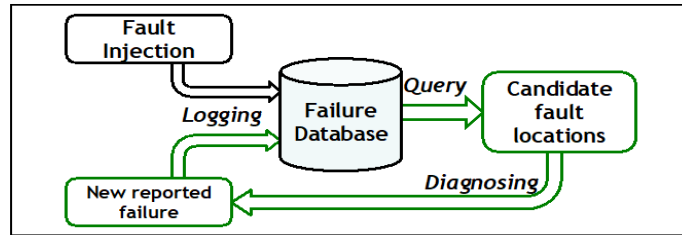
Table 6: CVE-2008-0600 DETECTOR /CRIU

Application	Runtime (s)	95% CI (s)	overhead (%)
F@H Normal	0.221	0.00922	0
F@H w/Detector	0.228	0.0122	3.30
F@H w/Naïve Detector	0.253	0.00851	14.4
pi-qmc Normal	0.137	0.00635	0
pi-qmc w/Detector	0.140	0.00736	1.73
pi-qmc w/Naïve Detector	0.152	0.00513	11.1

Failure diagnosis for distributed systems using targeted fault injection

We developed and evaluated a failure diagnosis technique that enables fast system and application fixes. Specifically, we propose a technique called Approximate Fault Localization, which allows fast determination of failure root-causes' locations and failure modes of distributed systems. We applied this technique in the context of OpenStack, an open source cloud management system, to reduce its maintenance cost.

The approach is summarized in Figure 24. When a failure is observed in a production system, we collect its failure profile through our distributed tracing tool. The collected failure profile is processed to reconstruct an end-to-end processing flow corresponding the failure. An end-to-end processing flow is a sequence of system events (e.g., system calls) across multiple distributed components that are invoked from the moment a request is received by the system, to the moment when the final response is returned, or the processing of the request is terminated (e.g., due to a failure). An example end-to-end flow of one request is given in Figure 24. This reconstructed end-to-end processing flow is then used to query against a pre-constructed Failure Profile Database to find faults that generate 'similar' flows. The returned faults are given to developers as hints to the locations in the processing flow where the actual root-caused fault might have occurred.

**Figure 24. Overview of the Approximate Fault Localization approach**

The approach is summarized in Figure 5. When a failure is observed in a production system, we collect its failure profile through our distributed tracing tool. The collected failure profile is processed to reconstruct an end-to-end processing flow corresponding the failure. An end-to-end processing flow is a sequence of system events (e.g., system calls) across multiple distributed components that are invoked from the moment a request is received by the system, to the moment when the final response is returned, or the processing of the request is terminated (e.g., due to a failure). An example end-to-end flow of one request is given in Figure 24. This reconstructed end-to-end processing flow is then used to query against a pre-constructed Failure Profile Database to find faults that generate 'similar' flows. The returned faults are given to

developers as hints to the locations in the processing flow where the actual root-caused fault might have occurred.

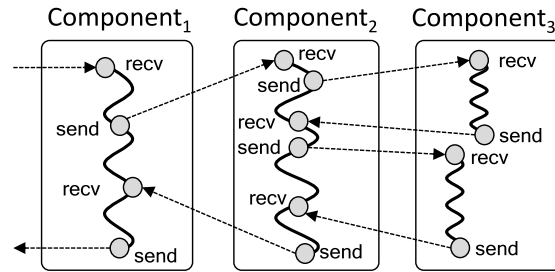


Figure 25. Example of an end-to-end flow

In order to construct the Failure Profile Database, we developed a fault injection framework, called Targeted Fault Injection, to allow deterministically injecting faults at exact locations in the execution flows of a distributed system. The fault injection framework consists of a specification language and a runtime system. The specification language is used to define precise fault injection scenarios. The runtime system takes the specification as input to generate execution plan and automate the fault injections. The collected profiles of fault-injected executions are added to the Failure Profile Database.

We evaluated the proposed approach with Open Stack to demonstrate its feasibility. We are trying to answer the following two questions. The results show that we can correctly determine the failed components, among multiple components of Open Stack, for 70-100% of tested cases.

Section 5 Conclusions

Cloud computing allows users to obtain scalable computing resources, but with a rapidly changing landscape of attack and failure modes, the effort to protect these complex systems is increasing. As we demonstrated VM monitoring plays an essential role in achieving resiliency. However, existing VM monitoring systems are frequently insufficient for cloud environments as those monitoring systems require extensive user involvement when handling multiple operating system (OS) versions. Cloud VMs can be heterogeneous, and therefore the guest OS parameters needed for monitoring can vary across different VMs and must be obtained in some way. Past work involves running code inside the VM, which may be unacceptable for a cloud environment.

We envisage that this problem will be solved by recognizing that there are common OS design patterns that can be used to infer monitoring parameters from the guest OS. We can extract information about the cloud user's guest OS with the user's existing VM image and knowledge of OS design patterns as the only inputs to analysis. As a proof of concept we have been developing VM monitors by applying this technique. Specifically, we implemented sample monitors that include a return-to-user attack detector and a process-based keylogger detector.

Another important aspect of delivering robust and efficient monitoring and protection against accidental failures and malicious attacks is our ability to validate (using formal and experimental methods) the detection capabilities of the proposed mechanisms and strategies. Towards that end we require development of validation frameworks that integrate the use of tools such as model

checkers (for formal analysis and symbolic execution of software) and fault/attack injectors (for experimental assessment).

Section 11 Trustworthiness Estimation for Workflow Completion (David Nicol & Jingwei Huang)

Section 1 Summary of Research Project

This research developed methods, models, and algorithms for trustworthiness estimation of cloud workflow, from the belief in the attributes of workflow components, and for trust judgment in a cloud service or a cloud entity, based on suitable trust mechanisms for the clouds.

Section 2 Introduction

We built trust models for clouds and cloud workflow, and developed a decision aid framework for workflow cloud resource management. We also attempted to estimate trustworthiness of Hadoop/YARN application (workflow of tasks) completion, by taking into account of not only performance attributes, but also security attributes.

The issues and challenges of trust in cloud computing have been widely discussed from different perspectives, and there is little research on formal models and systematical mechanisms of trust in clouds, or trust chains from users to services through various cloud entities. Hadoop has been widely deployed in cloud computing, and it is evolving through replacing MapReduce with YARN (the next generation); however, current research, design, and development of Hadoop focuses on the performance aspect of cloud, and the security issues and their impacts on Hadoop system are rarely studied.

Because of the criticality of many cloud services, cloud clients (especially organizations) need to make decisions about employing a cloud service based on “formal” trust, which is more certain, more accountable and more dependable than that based on some informal trust, such as experience and reputation as we often see in e-commerce. We explore such “formal” trust approaches for clouds, leveraging our experience with PKI trust mechanisms. In our trust mechanism design, the attributes of a cloud service (or cloud provider) that are critical to a cloud user are used as evidence for the user’s trust judgment, and the belief in those attributes is based on formal certification and chains of trust for validation. The proposed trust mechanism could meet the needs of cloud clients especially on mission-critical services. The mechanisms are based on past successful experience with modeling trust in PKI.

Trust itself is associated with risk, due to incomplete information. The more information and attributes we can know, the more we are confident about what we trust. We need to make our attribute-based calculus of trust be flexible to handle different level of information incompleteness and in different granularity.

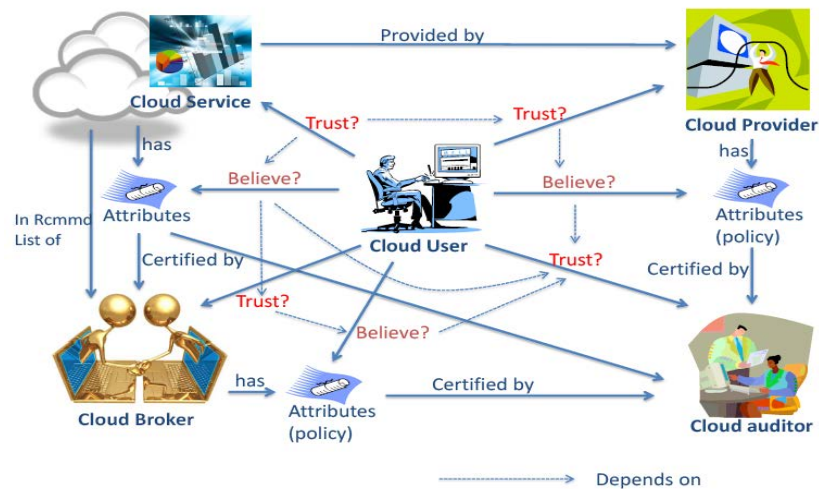


Figure 26. General structure for evidence-based trust judgement

- Based on the semantics of trust, we constructed a general structure for evidence-based trust judgment, which defines the attributes to be examined as evidence in a space of two-dimensions: domain of expectancy and source of trust including competence, goodwill, and integrity. Trust is a complex social phenomenon; this abstraction is based on the findings from social sciences.
- We developed workflow trust aggregation models, including sequence aggregation and parallel aggregation, and analyzed their properties. Different from our previous trust calculus, where trust propagates in a trust network; the new aggregation is from the perspective of workflow completion and “Reliability Block Graph” and has different operators.
- We developed a stochastic event simulation model that estimates the degree of belief of the completion of a workflow in Hadoop/YARN system, and studies the impacts of security issues to the workflow completion. In particular, we focus on the impacts of multi-tenancy and denial of service attack on Hadoop’s performance.

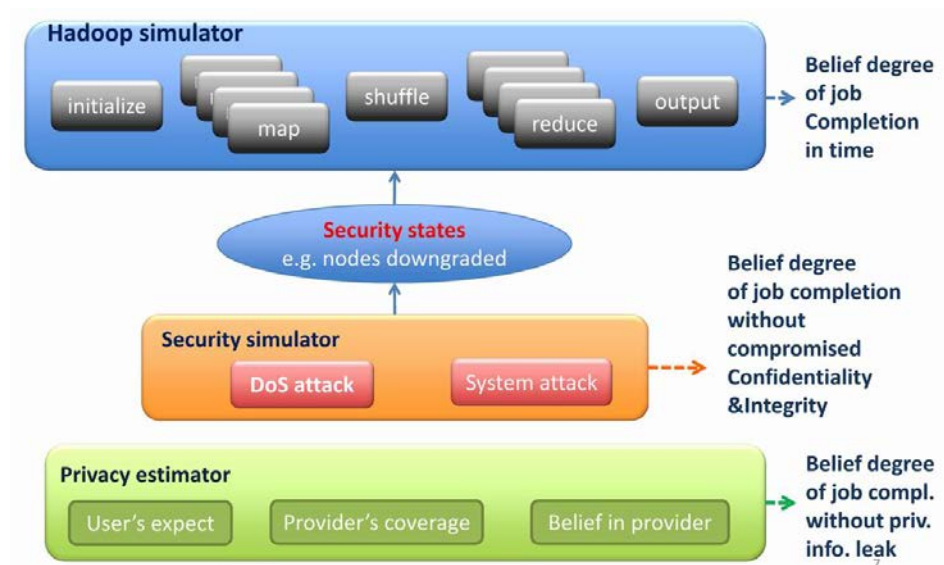


Figure 27. Hadoop/YARN system performance

Our simulation shows that even a small amount of compromised capacity may significantly degrade Hadoop/YARN system performance. The following figure shows the Cumulative Distribution Function of a MapReduce job completion time in some situations under denial of service attacks. (CCap denotes compromised capacity). For larger cluster, e.g. 10,000 nodes, the simulation results are similar.

- Hadoop cluster: 1000 nodes X 16GB; n=1000
- Job: 1024 mappers; 256 reducers; allocated capacity: 5%
- CCap: compromised capacity
- m: number of nodes compromised by DoS attack
- d: percentage of resources consumed by DoS attack
- S1: CCap=0
- S2: CCap=6.25% (m/n=78.78%), d=50%
- S3: CCap=6.25% (m/n=78.78%), d=80%
- S4: CCap=9.375% (m/n=99.95%), d=80%
- S5: CCap=9.375% (m/n=99.95%), d=90% -> cdf curve almost stays as 0
- S6: CCap=6.25% (m/n=78.78%), with longer maximum time limit for tasks

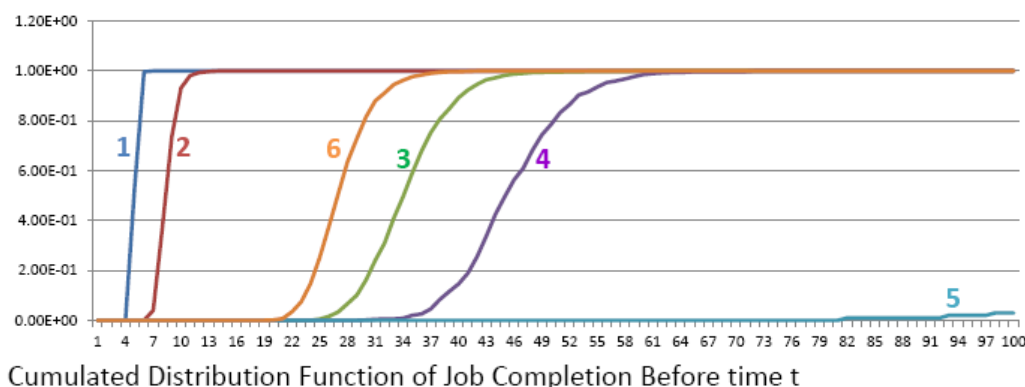


Figure 28. Cumulative Distribution Function of a MapReduce job completion time

Section 5 Conclusions

We developed a framework for using evidence-based trust to calculate the degree of belief in a service provider's performance with respect to satisfying a user's privacy protection expectation. Based on privacy theories, we identified users' expectation space and privacy policy compliance evidence space. Based on trust theories, we identified privacy CIA (Ability, Intension, Consistency) triad of evidence for trust. To infer trust from evidence, we constructed a specific form of Belief Network model, in which the uncertain belief in each variable (representing a proposition) is measured with a triple <belief degree, disbelief degree, unknown degree>; this model is an extension to the formal-semantics-based calculus of trust we previously developed. The framework is illustrated as the following figure.

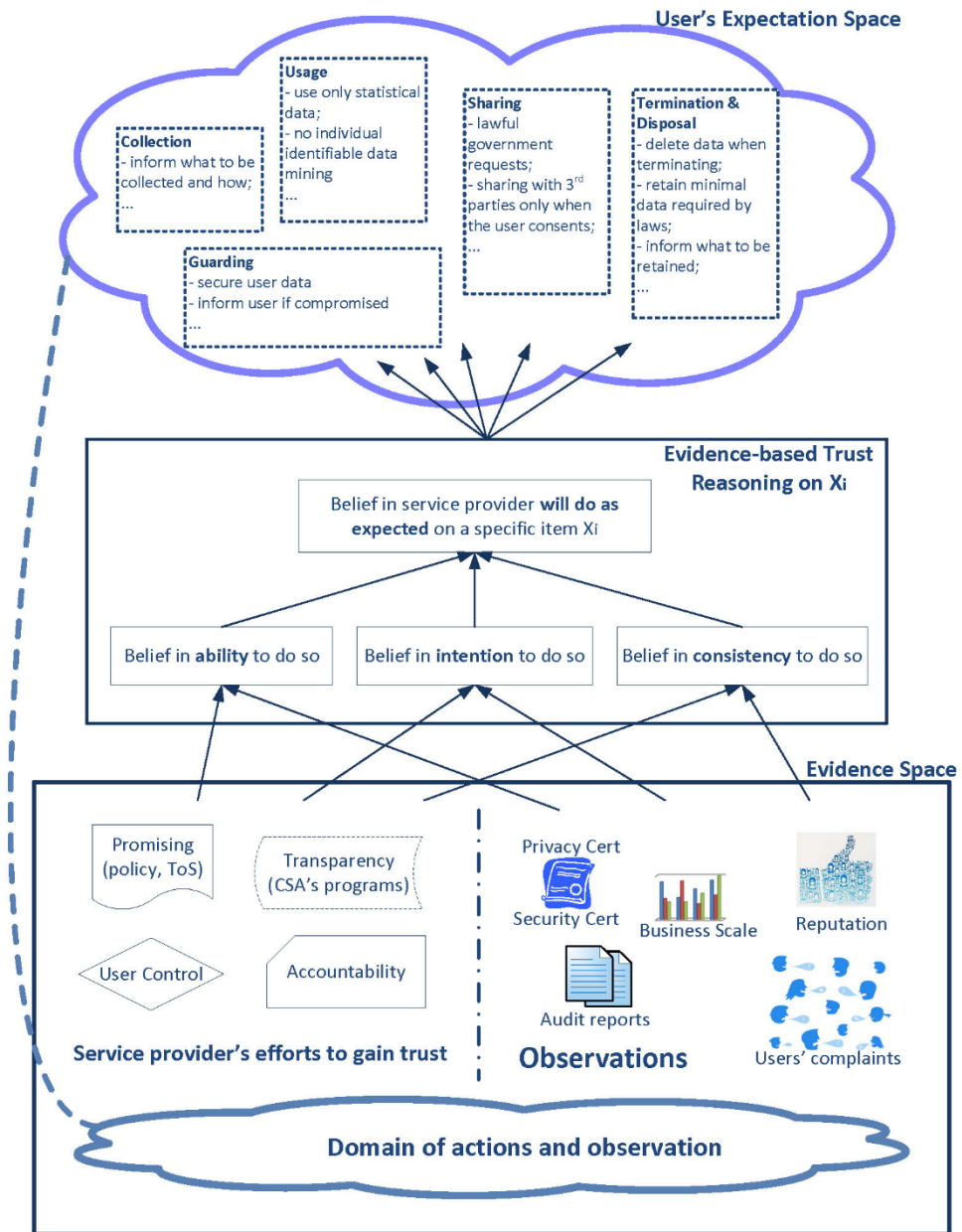


Figure 29. Belief Network model

Section 12 Application-Aware Cloud Network Resource Allocation (Roy Campbell, Chris Cai, and Gourav Kheneja)

Section 1 Summary of Research Project

This research is aimed at the performance and algorithms used in Cloud computing and addresses issues of network performance, geographic distribution and stream processing. The results of the research are encapsulated in the products of four separately titled thrusts:

- 1) Phurti: Application and Network-Aware Flow Scheduling for Multi-tenant MapReduce Cluster.
- 2) CRONets: Cloud-Routed Overlay Networks.
- 3) Data assurance in Clouds
- 4) Ambry: Geographically distributed blob store.
- 5) Samza: Large state in Stream Processing Systems.
- 6) R-Storm: Apache Storm is one of the most popular stream processing systems in industry today. However, Storm, like many other stream processing systems lacks an intelligent scheduling mechanism. R-Storm is designed to increase overall throughput by maximizing resource utilization while minimizing network latency.

Section 2 Introduction

Phurti: Phurti is a Mapreduce job traffic flow scheduling framework. Phurti uses an API to acquire the network topology knowledge from the application user. Given Phurti's understanding of the traffic characteristics of Mapreduce jobs, it schedules jobs so that the network traffic is considered at the job level instead of flow level.

CRONets: CRONets allows individual users to build their own overlay network using publicly available cloud servers, without explicit negotiation with ISPs. Taking advantage of TCIP's reliable transmission of data but its need for acknowledgments, CRONets optimizes transmission by reliably transmitting the data for partial segments of the route, avoiding high latency caused by error loss and retransmit.

Data assurance in Clouds: Availability of data stored in or across clouds is critical to the whole operation of the systems that depend on this data. Storage workload models that can be used to design better metadata management schemes and better placement schemes to achieve desired levels of availability.

SDN based policy enforcement: SDN networks are becoming more popular but require a controller and switches to interact. This research built SDN-based policy enforcement vis-à-vis network virtualization, address abstractions and work abstractions into a “Fabric” network using SDN-based label switching. The approach allows enclaves and virtualization of SDN network routes and controllers.

Ambry: Large social networks and similar Cloud applications allow users storage and sharing over large geographical areas. Latency and availability are key attributes of appropriate solutions. We researched a distributed storage system for storing large immutable objects (so called blobs) efficiently in geo-distributed environment.

Samza: This is research into handling very large state (100s of TBs to PBs) in stream processing systems which enables large joins and aggregations over streams. We developed large state handling on Apache Samza, and based on our evaluation we can reach up to 100x better performance compared to traditional way of handling state. Evaluated various state handling mechanisms and our proposed mechanism. Based on our results, our mechanism reaches low latency, high throughput, and almost constant failure recovery time. The research added the fault-tolerance to Apache Samza. Based on our evaluations, the failure recover can be in parallel and almost constant irrespective of the number of failures. We have design a mechanism to reduce failure recovery by preventing state rebuild as much as possible.

Examined the network as a resource in the context of graph processing, Supporting On-demand Elasticity in Distributed Graph Processing.

R-Storm: We evaluate R-Storm on set of microbenchmark Storm applications as well as Storm applications used in production at Yahoo! Inc. From our experimental results we conclude that R-Storm achieves 30-47% higher throughput and 69-350% better CPU utilization than default Storm for the micro-benchmarks. For the Yahoo! Storm applications, R-Storm out performs default Storm by around 50% based on overall throughput.

Section 3 Methods, Assumptions, and Procedures

Phurti: Phurti uses a centralized approach to gather both application information and network topology information through APIs. On top of Phurti framework, we develop a scheduling algorithm which prioritizes jobs with shortest sequential traffic.

CRONets: We build overlay network using cloud servers provided by IBM Softlayer at different geographical locations. We conduct extensive experiments using Planetlab located at a wide range of locations. We also examine the effect of Split-Overlay method to boost network

throughput. We also develop a new approach to dynamically choose the best overlay path out of multiple overlay paths.

Data assurance in Clouds: Analysis of real workloads. Workload modeling through statistical techniques. Validation through case studies in distributed storage systems.

SDN based policy enforcement: The SDN network routing was abstracted by an MLS routing scheme. The SDN Controller then mapped traffic through the network onto labels and transferred the traffic through MLS routes. The scheme allows virtualization of the routes, allowing policies to decide how to route within the SDN network.

Section 4 Results and Discussion

Phurti: Based on evaluation results using job traces sampled from Facebook cluster, Phurti improves job completion time for improves job completion time for 95% of the jobs, decreases average job completion time by 20% and tail job completion time by 13%.

CRONets: We show that CRONets improve the throughput for 78% of the default Internet paths with a median and average improvement factors of 1.67 and 3.27 times respectively, at a tenth of the cost of leasing private lines of comparable performance.

Data assurance in Clouds: Improved distributed storage system design with respect to metadata management, data placement, and availability in failure-prone scenarios. Demonstrations using real Big Data workloads from Yahoo!. Products include: Workload characterization, workload models, synthetic workload generator, and applications that demonstrate the usability of our models.

SDN based policy enforcement: We showed lightweight switching of routes and virtualization of controllers (hypervisor control for SDN). Switching was shown to retain integrity.

Ambry: In collaboration with LinkedIn, we have developed “Ambry”, a scalable geo-distributed object store. For over 2.5 years, Ambry has been the mainstream storage for all LinkedIn’s media objects, across all of its four datacenters, serving more than 450 million users. Our experimental results show that Ambry reaches high throughput (reaching up to 88% of the network bandwidth) and low latency (serving 1 MB objects in less than 50 ms), works efficiently across multiple geo-distributed datacenters, and improves the imbalance among disks by a factor of 8x-10x while moving minimal data.

Samza: In collaboration with LinkedIn, we have developed a scalable large state stream processing system. The results were:

The system is designed to handle very large state (100s of TBs to PBs) in stream processing systems which enables large joins and aggregations over streams. We developed large state

handling on Apache Samza, and based on our evaluation we can reach up to 100x better performance compared to traditional way of handling state. Based on our evaluations, the failure recover can be in parallel and almost constant irrespective of the number of failures. We have design a mechanism to reduce failure recovery by preventing state rebuild as much as possible. Evaluated various state handling mechanisms and our proposed mechanism. Based on our results, our mechanism reaches low latency, high throughput, and almost constant failure recovery time. Samza is a currently running system of Linked-In and the code is Open Source.

R-Storm: R-Storm (Resource-Aware Storm) is a system that implements resource-aware scheduling within Storm. R-Storm is designed to increase overall throughput by maximizing resource utilization while minimizing network latency. When scheduling tasks, R-Storm can satisfy both soft and hard resource constraints as well as minimizing network distance between components that communicate with each other. We evaluate R-Storm on set of micro-benchmark Storm applications as well as Storm applications used in production at Yahoo! Inc. From our experimental results we conclude that R-Storm achieves 30-47% higher throughput and 69-350% better CPU utilization than default Storm for the micro-benchmarks. For the Yahoo! Storm applications, R-Storm out performs default Storm by around 50% based on overall throughput. We also demonstrate that R-Storm performs much better when scheduling multiple Storm applications than default Storm.

Section 5 Conclusions

- 1) Network Aware Applications, Phurti, improves job completion time.
- 2) We have finished a set of prevalent measurement showing a significant portion of Internet paths could benefit from CRONets.
- 3) With CRONETS, we validate that using Split-Overlay perform better than plain overlay overlay for most cases.
- 4) The SDN based policy enforcement and hypervisor for virtual SDN network software is available for further experimentation.
- 5) We designed and developed an industry scale object store optimized for a geo-distributed environment (Ambry). Ambry serves requests in a geographically distributed environment of multiple datacenters while maintaining low latency and high throughput. Using a decentralized design, rebalancing mechanism, chunking, and logical blob grouping, we provide load balancing and horizontal scalability to meet the rapid growth at LinkedIn. Ambry source code available as Open Source contributions.
- 6) We minimized cross-datacenter traffic by using asynchronous writes (write to local datacenter and propagate to other in the background),
- 7) We designed and developed an industry scalable large state stream processing system (Samza). We developed a 2 phase background replication mechanism. We developed a load balancing mechanism returning the system to a balanced state after expansion.
- 8) Ambry and Samza are in current use by Linked-In
- 9) R-Storm out performs default Storm by around 50% based on overall throughput

2017

1. Roy H. Campbell, Charles Kamhoua, and Kevin Kwiat (Eds.), **Assured Cloud Computing**, IEEE-Wiley, New York, NY, to be published 2017.
2. Shadi A. Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringham, Indranil Gupta, and Roy H. Campbell, "Samza: Stateful Stream Processing at Scale", *43rd International Conference on Very Large Data Bases (VLDB 2017)*, Munich, Germany, August 28-September 1, 2017, to appear.
3. Carlo Di-Giulio, Read Sprabery, Charles Kamhoua, Kevin Kwiat, Roy Campbell, and Masooda Bashir, "Cloud Standards in Comparison: Are New Security Frameworks Improving Cloud Security?", *10th IEEE International Conference on Cloud Computing (Cloud 2017)*, Honolulu, HI, June 25-June 30, 2017, to appear.
4. Carlo Di-Giulio, Read Sprabery, Charles Kamhoua, Kevin Kwiat, Roy Campbell, and Masooda Bashir, "IT Security and Privacy Standards in Comparison: Improving FedRAMP Authorization for Cloud Service Providers", *International Workshop on Assured Cloud Computing and QoS Aware Big Data (WACC 2017)*, Madrid, Spain, May 14, 2017.
5. Zackary J. Estrada, Read Sprabery, Lok Yan, Zhongzhi Yu, Roy Campbell, Zbigniew Kalbarczyk, and Ravishankar K. Iyer, "Using OS Design Patterns to Provide Reliability and Security as-a-Services for VM-based Clouds", *13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2017)*, Xi'an China, April 8-9, 2017.
6. Read Sprabery, Zachary Estrada, Jon Calhoun, Zbigniew Kalbarczyk, Ravishankar Iyer, Rakesh B. Bobba, and Roy Campbell, "Trustworthy Services Built on Event Based Probing for Layered Defense", *IEEE International Conference on Cloud Engineering (IC2E 2017)*, Vancouver, Canada, April 4-7, 2017.
7. Phuong Cao, Alexander Withers, Zbigniew Kalbarczyk, and Ravishankar Iyer, "Learning Factor Graphs for Preempting Multi-State Attacks in Cloud Infrastructure", poster, *Symposium and Bootcamp on the Science of Security (HotSoS 2017)*, Hanover, MD, April 4-5, 2017.
8. Stephen Skeirik, Andrei Stefanescu, and Jose Meseguer, "A Constructor-Based Reachability Logic for Rewrite Theories", Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, March 27, 2017.
9. Carlo Di-Giulio, Read Sprabery, Charles Kamhoua, Kevin Kwiat, Roy Campbell, and Masooda Bashir, "Cloud Security Certifications: A Comparison to Improve Cloud Service Provider Security", *International Conference on Internet of Things, Data and Cloud Computing (ICC 2017)*, Churchill College, Cambridge, UK, March 22-23, 2017.
10. SI Liu, Jatin Ganhotra, Muntasir Raihan Rahman, Son Nguyen, Indranil Gupta, and Jose Meseguer, "Quantitative Analysis of Consistency in NoSQL Key-value Stores", *Leibniz Transactions on Embedded Systems (LITES Special Issue on Quantitative Evaluation of Systems (QEST))*, volume 4, number 1, 2017.

2016

1. Read Sprabery, Zachary Estrada, Zbigniew Kalbarczyk, Ravishankar Iyer, Roy Campbell, and Rakesh Bobba, "Defense in Depth for Virtual Applications Built on Event Based Probing of

- Untrusted Guests,” *Annual Computer Security Applications Conference (ACSAC 2016)*, Los Angeles, CA, December 5-9, 2016.
2. Read Sprabery, Güliz Seray Tuncay, Carl Gunter, and Roy Campbell, “Securely Retrofitting Door Locks for Cheap Control through Mobile Devices”, *Annual Computer Security Applications Conference (ACSAC 2016)*, Los Angeles, CA, December 5-9, 2016.
 3. Marjan Sirjani, Ehsan Khamespanah, Kirill Mechitov and Gul Agha, “A Compositional Approach for Modeling and Timing Analysis of Wireless Sensor and Actuator Networks”, *9th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2016)*, Porto, Portugal, November 29, 2016.
 4. Keywhan Chung, Valerio Formicola, Alexander Withers, Adam Slagell, Zbigniew Kalbarczyk, and Ravishankar Iyer, “Attacking Supercomputers Through Targeted Alteration of Environmental Control: A Data Driven Case Study,” *International Workshop on Cyber-Physical Systems Security*, in conjunction with *IEEE Conference on Communications and Network Security (CNS 2016)*, Philadelphia, PA, October 17-19, 2016.
 5. Ahmed Fawaz, Atul Bohara, Carmen Cheh, and William H. Sanders, “Lateral Movement Detection using Distributed Data Fusion”, *35th Symposium of Reliable Distributed Systems (SRDS 2016)*, Budapest, Hungary, September 26-29, 2016.
 6. Peter Olveczky, “Design and Validation of the P-Store Replicated Data Store in Maude”, *23rd International Workshop on Algebraic Development Techniques (WADT 2016)*, Gregynog, Wales, September 21-24, 2016.
 7. Shadi A. Noghabi, Roy H. Campbell, and Indranil Gupta, “Building a Scalable Distrusted Online Media Processing Environment”, *42nd International Conference on Very Large Data Bases (VLDB 2016)*, New Delhi, India, September 5-9, 2016.
 8. Muntasir Raihan Rahman, Lewis Tseng, Son Nguyen, Indranil Gupta, and Nitin Vaidya, “Characterizing and Adapting the Consistency-Latency Tradeoff in Distributed Key-value Stores”, *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, volume 11, issue 3, September 2016.
 9. Peter Csaba Olveczky, “Design and Validation of Cloud Computing Data Stores using Formal Methods”, invited paper, *International Symposium on Intelligent Systems and Applications (ISA 2016)*, Ho Chi Minh City, Vietnam, August 22-23, 2016.
 10. Gul Agha, “Abstractions, Semantic Models and Analysis Tools for Concurrent Systems: Progress and Open Problems”, *14th International Conference on Software Engineering and Formal Methods (SEFM 2016)*, Vienna, Austria, July 4-8, 2016.
 11. Uttam Thakore, Gabriel A. Weaver, and William H. Sanders, “A Quantitative Methodology for Security Monitor Deployment”, *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*, Toulouse, France, June 28-July 1, 2016. **Best Paper Award.**
 12. Chris X. Cai, Franck Le, Xin Sun, Geoffrey Xi, Hani Jamjoun, and Roy H. Campbell, “CRONets: Cloud-Routed Overlay Networks”, *36th IEEE International Conference on Distributed and Computing Systems (ICDCS 2016)*, Nara, Japan, June 27-30, 2016.
 13. Shadi A. Noghabi, Sriram Subramanian, Priyesh Narayanan, Sivabalan Narayanan, Gopalakrishna Holla, Mammad Zadeh, Tianwei Li, Indranil Gupta, and Roy H. Campbell, “Ambry: LinkedIn’s Scalable Geo-Distributed Object Store”, *2016 ACM SIGMOD/PODS*, San Francisco, CA, June 26 - July 1, 2016.
 14. Atul Bohara, Uttam Thakore, and William H. Sanders, “Intrusion Detection in Enterprise Systems by Combining and Clustering Diverse Monitor Data”, *Symposium and Bootcamp on the Science of Security (HotSoS 2016)*, Pittsburgh, PA, April 20-21, 2016.
 15. Ehsan Khamespanah, Kirill Mechitov, Marjan Sirjani, and Gul Agha, “Schedulability Analysis of Distributed Real-Time Sensor Network Applications using Actor-based Model Checking”, *23rd International SPIN Symposium on Model Checking of Software (SPIN 2016)*, Eindhoven, The Netherlands, April 7-8, 2016.

16. Mayank Pundir, Manoj Kumar, Luke Leslie, Indranil Gupta, and Roy H. Campbell, "Supporting On-demand Elasticity in Distributed Graph Processing", *IEEE International Conference on Cloud Engineering (IC2E 2016)*, Berlin, Germany, April 4-8, 2016. **Best Paper Award.**
17. Chris X. Cai, Shayan Saeed, Indranil Gupta, Roy Campbell, and Franck Le, "Phurti: Application and Network-Aware Flow Scheduling for Multi-tenant MapReduce Cluster", *IEEE International Conference on Cloud Engineering (IC2E 2016)*, Berlin, Germany, April 4-8, 2016.
18. Le Xu, Boyang Peng, and Indranil Gupta, "Stela: Enabling Stream Processing Systems to Scale-in and Scale-out On-Demand", *IEEE International Conference on Cloud Engineering (IC2E 2016)*, Berlin, Germany, April 4-8, 2016.
19. Si Liu, Peter Olveczky, Muntasir Raihan Rahman, Jatin Ganhotra, Indranil Gupta, and Jose Meseguer, "Formal Modeling and Analysis of RAMP Transaction Systems", *31st ACM Symposium on Applied Computing (SAC 2016)*, Pisa, Italy, April 4-8, 2016.
20. Minas Charalambides, Peter Dinges, and Gul Agha, "Parameterized, Concurrent Session Types for Asynchronous Multi-Actor Interactions", *Science of Computer Programming*, volume 115-116, pages 100-126, January-February 2016.
21. Keywhan Chung, Charles A. Kamhoua, Kevin A. Kwiat, Zbigniew T. Kalbarczyk and Ravishankar K. Iyer, "Game Theory with Learning for Cyber Security Monitoring", *17th IEEE International Symposium on High Assurance Systems Engineering (HASE 2016)*, Orlando, FL, January 7-9, 2016.

2015

1. Fangzhou Yao, Kevin Chen-Chuan Chang, and Roy H. Campbell, "Ushio: Analyzing News Media and Public Trends in Twitter", *8th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2015)*, Limassol, Cyprus, December 7-10, 2015.
2. Boyang Peng, Mohammad Hosseini, Zhihao Hong, Reza Farivar, and Roy Campbell, "R-Storm: Resource-Aware Scheduling in Storm", *Middleware 2015*, Vancouver, Canada, December 7-11, 2015.
3. Mainak Ghosh, Wenting Wang, Gopalakrishna Holla, and Indranil Gupta, "Morphus: Supporting Online Reconfigurations in Sharded NoSQL Systems", *IEEE Transactions on Emerging Topics in Computing*, volume PP, issue 99, November 11, 2015.
4. Weijie Liu, Rakesh B. Bobba, Sibin Mohan, and Roy H. Campbell, "Inter-Flow Consistency: A Novel SDN Update Abstraction for Supporting Inter-Flow Constraints", *IEEE Conference on Communications and Network Security (CNS 2015)*, Florence, Italy, September 28-30, 2015.
5. Jay P. Kesan, Carol Mullins Hayes, and Masooda Bashir, "Shaping Privacy Law and Policy by Examining the Intersection of Knowledge and Opinions", *Research Conference on Communication, Information and Internet Policy (TPRC 43)*, Arlington, VA, September 25-27, 2015.
6. Yosub Shin, Mainak Ghosh, and Indranil Gupta, "Parqua: Online Reconfigurations in Virtual Ring-Based NoSQL Systems", *IEEE International Conference on Cloud and Autonomic Computing (ICAC 2015)*, San Diego, CA, September 17-21, 2015.
7. Zachary J. Estrada, Cuong Pham, Fei Deng, Lok Yan, Zbigniew Kalbarczyk, and Ravishankar K. Iyer, "Dynamic VM Dependability Monitoring Using Hypervisor Probes", *11th European Dependable Computing Conference (EDCC 2015)*, Paris, France, September 7-11, 2015.
8. Si Liu, Son Nguyen, Jatin Ganhotra, Muntasir Raihan Rahman, Indranil Gupta, and José Meseguer, "Quantitative Analysis of Consistency in NoSQL Key-value Stores", *12th International Conference on Quantitative Evaluation of SysTems (QEST 2015)*, Madrid, Spain, September 1-3, 2015. **Nominee for Best Paper Award**

9. Mayank Pundir, Luke M. Leslie, Indranil Gupta, and Roy H. Campbell, "Zorro: Zero-Cost Reactive Failure Recovery in Distributed Graph Processing", *ACM Symposium on Cloud Computing (SoCC 2015)*, Kohala Coast, Hawaii, August 27-29, 2015.
10. Gary Wang, Zachary J. Estrada, Cuong Pham, Zbigniew Kalbarczyk, and Ravishankar K. Iyer, "Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring," to appear *9th USENIX Workshop on Offensive Technologies (WOOT 2015)*, Washington, DC, August 10-11, 2015.
11. Reza Shiftehfar, Kirill Mechitov and Gul Agha, "A Fine-Grained Adaptive Middleware Framework for Parallel Mobile Hybrid Cloud Applications", *6th Annual International Conference on ICT: Big Data, Cloud and Security*, Singapore, July 27-28, 2015. **Best paper award.**
12. Mainak Ghosh, Wenting Wang, Gopalakrishna Holla, and Indranil Gupta, "Morphus: Supporting Online Reconfigurations in Sharded NoSQL Key-value Stores", *12th IEEE International Conference on Autonomic Computing (ICAC 2015)*, Grenoble, France, July 7-10, 2015.
13. Mainak Ghosh, Indranil Gupta, Shalmoli Gupta, and Nirman Kumar, "Fast Compaction Algorithms for NoSQL Databases", *35th IEEE International Conference on Distributed Computing Systems (ICDCS 2015)*, Columbus, OH, June 29-July 2, 2015.
14. Phuong Cao, Eric Badger, Alexander Withers, Adam Slagell, Zbigniew Kalbarczyk, and Ravishankar Iyer, "Towards an Unified Security Testbed and Security Analytics Framework", *Symposium and Bootcamp on the Science of Security (HotSoS 2015)*, April 21-22, 2015.
15. Phuong Cao, Eric Badger, Adam Slagell, Zbigniew Kalbarczyk, and Ravishankar Iyer, "Preemptive Intrusion Detection: Theoretical Framework and Real-world Measurements", *Symposium and Bootcamp on the Science of Security (HotSoS 2015)*, April 21-22, 2015.
16. Weijie Liu, Rakesh B. Bobba, Sibin Mohan, and Roy H. Campbell, "Inter-Flow Consistency: Novel SDN Update Abstraction for Supporting Inter-Flow Constraints", *NDSS Workshop on Security of Emerging Networking Technologies (SENT)* co-located with *Network and Distributed System Security Symposium (NDSS 2015)*, San Diego, CA, February 8, 2015.

2014

1. Fangzhou Yao and Roy H. Campbell, "SafeBox: SCADA Systems in a Secure Framework", *5th Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS)*, Rome, Italy, December 2-5, 2014.
2. Peter Dinges and Gul Agha, "Solving Complex Path Conditions through Heuristic Search on Induced Polytopes", *22nd ACM SIGSOFT Symposium on Foundations of Software Engineering*, Hong Kong, November 16-21, 2014.
3. Si Liu, Muntasir, Raihan Rahman, Stephen Skeirik, Indranil Gupta, and Jose Meseguer, "Formal Modeling and Analysis of Cassandra in Maude", *International Conference in Formal Engineering Methods (ICFEM 2014)*, Luxembourg, November 3-7, 2014.
4. Abhishek Verma, Ludmila, Cherkasova and Roy H. Campbell, "Profiling and Evaluating Hardware Choices for MapReduce Environments: an Application-Aware Approach", *32nd International Symposium on Computer Performance, Modeling, Measurements, and Evaluation (IFIP WG 7.3 Performance 2014)*, Turin, Italy, October 7-9, 2014.
5. Cuong Pham, Zachary Estrada, Phuong Cao, Zbigniew Kalbarczyk, and Ravishankar Iyer, "Building Reliable and Secure Virtual Machines using Architectural Invariants", *IEEE Security and Privacy Magazine*, volume 12, issue 5, September-October 2014.
6. Peter Dinges and Gul Agha, "Targeted Test Input Generation using Symbolic-concrete Backward Execution" *29th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Västerås, Sweden, September 15-19, 2014.

7. Carol Mullins Hayes, Jay P. Kesan, Masooda Bashir, Kevin Hoff, and Gahyun Jeon, "Knowledge, Behavior, and Opinions Regarding Online Privacy", *Research Conference on Communication, Information and Internet Policy (TPRC 42)*, Arlington, VA, September 25-27, 2014.
8. Kevin Hoff and Masooda Bashir, "Trust in Automation: Integrating Empirical Evidence on Factors That Influence Trust", *Human Factors: The Journal of the Human Factors and Ergonomics Society*, volume 57, issue 3, September 2, 2014.
9. Jon Grov and Peter Csaba Olveczky, "Increasing Consistency in Multi-site Data Stores: Megasotre-CGC and Its Formal Analysis", *12th International Conference on Software Engineering and Formal Methods (SEFM 2014)*, Grenoble, France, September 1-5, 2014.
10. Wojciech Golab, Muntasir Raihan Rahman, Alvin Auyoung, Kimberly Keeton, Indranil Gupta, "Client-centric Benchmarking of Eventual Consistency for Cloud Storage Systems", *IEEE International Conference on Distributed Computing Systems (ICDCS 2014)*, Madrid, Spain, June 30-July 3, 2014.
11. Fangzhou Yao and Roy H. Campbell, "CouchFS: A High-Performance File System for Large Data Sets", *3rd International Congress on Big Data (BigData 2014)*, Anchorage, AK, June 27-July 2, 2014.
12. Fangzhou Yao and Roy H. Campbell, "CryptVMI: Encrypted Virtual Machine Introspection in the Cloud" *7th IEEE International Conference on Cloud Computing (IEEE Cloud 2014)*. Anchorage, AK, June 27-July 2, 2014.
13. Jingwei Huang, Zbigniew Kalbarczyk, and David M. Nicol, "Knowledge Discovery from Big Data for Intrusion Detection Using LDA", *3rd International Congress on Big Data (BigData 2014)*, Work-in-Progress Track paper, Anchorage, AK, June 26 – July 2, 2014.
14. Jingwei Huang, David M. Nicol, and Roy H. Campbell, "Denial-of-Service Threat to Hadoop/YARN Clusters with Multi-Tenancy", *3rd International Congress on Big Data (BigData 2014)*, Anchorage, AK, June 26-July 2, 2014.
15. Reza Shiftehfar, Kirill Mechitov, and Gul Agha, "Towards a flexible fine-grained access control system for mobile cloud applications", *7th IEEE International Conference on Cloud Computing (IEEE Cloud 2014)*, Work-in-Progress Track paper, Anchorage, AK, June 26-July 2, 2014.
16. Gul Agha, "Actors Programming for the Mobile Cloud." *13th International Symposium on Parallel and Distributed Computing (ISPDC 2014)*, Porquerolles Island, Cote d’Azur, France, June 24-27, 2014.
17. Cuong Pham, Zbigniew Kalbarczyk, and Ravishankar K. Iyer, "Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants", *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*, Atlanta, GA, June 23-26, 2014. **DSN Best Paper Award.**
18. Cuong Pham, Zachary Estrada, Phuong Cao, Zbigniew Kalbarczyk, and Ravishankar K. Iyer "HyperTap: Security Monitoring for Virtual Machines Using Hardware Architectural Invariants", *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*, Atlanta, GA, June 23-26, 2014.
19. R. Ramamurthy, Z. Estrada, C. Pham, Z. Kalbarczyk, and R. Iyer, "Designing a Performance Isolation Benchmark for Virtualized Systems", *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*, Fast Abstract, Atlanta, GA, June 23-26, 2014.

20. Gary Wang, Zachary Estrada, Cuong Pham, Zbigniew Kalbarczyk, Ravishankar Iyer, "Hypervisor Introspection: Exploiting Timing Side-Channels against VM Monitoring", *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*, Fast Abstract, Atlanta, GA, June 23-26, 2014.
21. Cristina L. Abad, Yi Lu, Roy H. Campbell and Nathan Roberts "A Model-Based Namespace Metadata Benchmark for HDFS", *USENIX International Conference on Autonomic Computing*, Philadelphia, PA, June 17-20, 2014.
22. Fangzhou Yao, Read T. Spraybery and Roy H. Campbell, "CryptVMI: a Flexible and Encrypted Virtual Machine Introspection in the Cloud", *2nd International Workshop on Security in Cloud Computing*, Kyoto, Japan, June 3-6, 2014.
23. Jingwei Huang and David M. Nicol. "Evidence-based Trust Reasoning", *Symposium and Bootcamp on the Science of Security (HotSoS 2014)*, Raleigh, NC, April 8-9, 2014.
24. Phuong Cao, Key-whan Chung, Adam Slagell, Zbigniew Kalbarczyk, and Ravishankar Iyer, "Preemptive Intrusion Detection", *Symposium and Bootcamp on the Science of Security (HotSoS 2014)*, Raleigh, NC, April 8-9, 2014.
25. S. Baset, L. Wang B. Tak, C. Pham, and C.Q. Tang, "Toward Achieving Operational Excellence in a Cloud", *IBM Journal of Research and Development, Issue topic on Software-Defined Environment*, volume 58, issue 2/3, March-May 2014.
26. Cuong Manh Pham, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Victor Dogaru, Rohit Wagle, Chitra Venkatramani, "An Evaluation of ZooKeeper For High Availability in System S," *5th ACM/SPEC International Conference on Performance Engineering*, Dublin, Ireland March 22-26, 2014.
27. Furquan Shaikh, Fangzhou Yao, Indranil Gupta and Roy H. Campbell, "VMDedup: Memory Deduplication in Hypervisor", *IEEE International Workshop on Cloud Analytics (IWCA 2014)*, Boston, MA, March 11, 2014.
28. YoungMin Kwon, Kirill Mechitov, and Gul Agha, "Design and Implementation of a Mobile Actor Platform for Wireless Sensor Networks", *Concurrent Objects and Beyond*, pp. 276-316. *Lecture Notes in Computer Science*, Volume 8665, 2014, pages 276-316, 2014.
29. Jon Grov and Peter Csaba Olveczky, "Formal Modeling and Analysis of Google's Megastore in Real-Time Maude", *Specification, Algebra, and Software – Essays Dedicated to Kokichi Futatsugi, Lecture Notes in Computer Science*, volume 8373, pages 494-519, 2014.

2013

1. Uttam Thakore, Gabriel A. Weaver, and William H. Sanders, "An Actor-Centric, Asset-Based Monitor Deployment Model for Cloud Computing", *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013)*, Dresden, Germany, December 9-12, 2013.
2. Imranul Hoque and Indranil Gupta, "LFGraph: Simple and Fast Distributed Graph Analytics", *ACM Symposium on Timely Results in Operating Systems (TRIOS 2013)*, Farmington, PA, November 3, 2013.
3. Brian Cho, Muntasir Rahman, Tej Chajed, Indranil Gupta, Cristina Abad, Nathan Roberts, Philbert Lin, "Natjam: Design and Evaluation of Eviction Policies for Supporting Priorities and Deadlines in Mapreduce Clusters", *4th Annual Symposium on Cloud Computing (SoCC 2013)*, Santa Clara, CA, October 1-3, 2013.

4. Cristina L. Abad, Mindi Yuan, Chris X. Cai, Yi Lu, Nathan Roberts, and Roy H. Campbell; "Generating Request Streams on Big Data using Clustered Renewal Processes", *Performance Evaluation Journal, Proceedings of the IFIP Performance 2013*, volume 70, issue 10, October 2013.
5. Cuong Pham, Qingkun Li, Zachary Estrada, Zbigniew Kalbarczyk, and Ravishankar K. Iyer, "A Simulation Framework to Evaluate Virtual CPU Scheduling Algorithms", *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshop (ICDCSW 2013)*, Philadelphia, PA, July 8-11, 2013.
6. Martin Vigil, Daniel Cabarcas, Jingwei Huang, and Johannes Buchmann, "Assessing Trust in the Long-Term Protection of Documents", *18th IEEE Symposium on Computers and Communications (ISCC 2013)*, Split, Croatia, July 7-10, 2013.
7. Youngmin Kwon and Gul Agha, "Performance Evaluation of Sensor Networks by Statistical Modeling and Euclidean Model Checking", *ACM Transactions on Sensor Networks (TOSN)* volume 9, issue 4, article number 39, July 2013.
8. Chris X. Cai, Cristina L. Abad, and Roy H. Campbell; "Storage-Efficient Data Replica Number Computation for Multi-level Priority Data in Distributed File Systems", *Workshop on Reliability and Security Data Analysis (RSDA 2013)*, co-located with the *43rd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013)*, Budapest, Hungary, June 24-27, 2013.
9. Salman Malik, Mirko Montanari, Jun Ho Huh, Rakesh B. Bobba, and Roy H. Campbell, "Towards SDN Enabled Network Control Delegation in Clouds", *Third International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV 2013)*, co-located with the *43rd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013)*, Budapest, Hungary, June 24-27, 2013.
10. Peter Dinges, Minas Charalambides, and Gul Agha, "Automated inference of atomic sets for safe concurrent execution", *11th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2013)*, Seattle, WA, June 20, 2013.
11. Mirko Montanari, Jun Ho Hun, Rakesh B. Bobba, and Roy H. Campbell, "Limiting Data Exposure in Monitoring Multi-domain Policy Conformance", *6th International Conference on Trust and Trustworthy Computing (TRUST 2013)*, London, UK, June 17-19, 2013.
12. Jun Ho Huh, Mirko Montanari, Derek Dagit, Rakesh Bobba, Dong Wook Kim, Yoonjoo Choi and Roy H Campbell, "An Empirical Study on the Software Integrity of Virtual Appliances: Are You Really Getting What You Paid For?", *8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS 2013)*, Hangzhou, China, May 13-16, 2013.
13. Stephen Skeirik, Rakesh B. Bobba, and Jose Meseguer, "Formal Analysis of Fault-tolerant Group Key Management using ZooKeeper", *First International Workshop on Assured Cloud Computing Conference (CCGrid 2013)*, *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Delft, The Netherlands, May 13-16, 2013.
14. Jingwei Huang and David M. Nicol, "Trust Mechanisms for Cloud Computing", *Journal of Cloud Computing*, volume, 2 issue 9, April 2013.
15. Jun Ho Huh, Mirko Montanari, Derek Dagit, Rakesh Bobba, Dong Wook Kim, Yoonjoo Choi and Roy H Campbell, "Assessing Software Integrity of Virtual Appliances through Software Whitelists: Is it any good?", *Network & Distributed System Security Symposium (NDSS 2013)*, San Diego, CA, February 24-27, 2013.

2012

1. Faraz Faghri, Sobir Bazarbayev, Mark Overholt, Reza Farivar, Roy H. Campbell, and William H. Sanders, "Failure Scenario as a Service (FSaaS) for Hadoop Clusters," *Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, in conjunction with the 13th *International Conference on Middleware (Middleware 2012)*, Montreal, Quebec, Canada, December 3-7, 2012.
2. Cristina Abad, Huong Luu, Nathan Roberts, Kihwal Lee, Yi Lu, and Roy H. Campbell; "Metadata Traces and Workload Models for Evaluating Big Storage Systems," *2012 IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2012)*, Chicago, IL, November 5-8, 2012.
3. Cristina Abad, Nathan Roberts, Yi Lu, and Roy H. Campbell, "A Storage-Centric Analysis of MapReduce Workloads: File Popularity, Temporal Locality and Arrival Patterns", *2012 IEEE International Symposium on Workload Characterization (IISWC 2012)*, La Jolla, CA, November 4-6, 2012.
4. Jingwei Huang and David M. Nicol, "Security and Provenance in M3GS for Cross-domain Information Sharing", *IEEE Military Communications Conference (MILCOM 20112)*, Orlando, FL, October 29-November 1, 2012.
5. Ralf Sasse, Samuel T. King, Jose Meseguer, and Shuo Tang, "IBOS: A Correct-By-Construction Modular Browser", *9th International Symposium on Formal Aspects of Component Software (FACS 2012)*, Mountain View, CA, September 12-14, 2012.
6. Minas Charalambides, Peter Dinges, and Gul Agha, "Parameterized Concurrent Multi-Party Session Types", *111th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems (FOCLASA 2012)*, New Castle, United Kingdom, September 8, 2012.
7. Mirko Montanari, Lucas T. Cook, and Roy H. Campbell, "Multi-organization Policy-based Monitoring", *2012 IEEE Policies for Distributed Systems and Networks (POLICY 2012)*, Chapel Hill, NC, July 16-18, 2012.
8. Cuong Pham; Phuong Cao; Zbigniew Kalbarczyk, and Ravishankar K. Iyer, "Toward a High Availability Cloud: Techniques and Challenges", *2nd International Workshop on Dependability of Clouds, Data Centers, and Virtual Machine Technology*, in conjunction with 42nd *International IEEE/IFIP Conference on Dependable Systems and Networks (DSN 2012)*, Boston, MA, June 25-28, 2012.
9. Catello Di Martino, Marcello Cinque, and Domenico Cotroneo, "Assessing Time Coalescence Techniques for the Analysis of Supercomputer Logs", *42nd International IEEE/IFIP Conference on Dependable Systems and Networks (DSN 2012)*, Boston, MA, June 25-28, 2012.
10. Mirko Montanari, Jun Ho Huh, Derek Dagit Rakesh Bobba and Roy H. Campbell, "Evidence of Log Integrity in Policy-based Security Monitoring", *2nd International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV 2012)*, in conjunction with the 42nd *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, Boston, MA, June 25-28, 2012.
11. Jingwei Huang, and David M. Nicol, Rakesh Bobba, and Jun Ho Huh, "A Framework Integrating Attribute-based Policies into Role Based Access Control", *17th ACM Symposium on Access Control Models and Technologies (SACMAT 2012)*, Newark, NJ, June 20-22, 2012.
12. Peter Dinges and Gul Agha, "Scoped Synchronization Constraints for Large Scale Actor Systems", *COORDINATION 2012*, Stockholm, Sweden, June 14-15, 2012.

13. Lucian Bentea and Peter Olveczky, "A Probabilistic Strategy Language for Probabilistic Rewrite Theories and its Application to Cloud Computing", *21st International Workshop on Algebraic Development Trends (WADT 2012)*, Salamanca, Spain, June 7-10, 2012.
14. Abhishek Verma, Ludmila Cherkasova, Vijay Kumar, and Roy H. Campbell, "Deadline-based Workload Management for MapReduce Environments: Pieces of the Performance Puzzle", *2012 IEEE/IFIP Network Operations Management Symposium (NOMS 2012)*, Maui, HI, April 16-20, 2012.
15. Jonas Eckhardt, Tobias Muhlbauer, Musab Al-Turki, Jose Meseguer, and Martin Wirsing, "Stable Availability under Denial of Service Attacks through Formal Parameters", *15th International Conference Fundamental Approaches to Software Engineering (FASE 2012)*, Tullin, Estonia, March 24-April 1, 2012.
16. Martin Wirsing, Jonas Eckhardt, Tobias Muhlbauer and Jose Meseguer, "Design and Analysis of Cloud-Based Architectures with KLAIM and Maude", *15th International Conference on Fundamental Approaches to Software Engineering (FASE 2012)*, Tullin, Estonia, March 24-April 1, 2012.

2011

1. Masooda Bashir, Jay P. Kesan, Carol M. Hayes, and Robert Zielinski; "Privacy in the Cloud: Going Beyond the Contractarian Paradigm", *2011 Workshop on Governance of Technology, Information, and Policies (GTIP 2011)*, Orlando, FL, December 6, 2011.
2. Roy H. Campbell, Mirko Montanari, and Reza Farivar; "A Middleware for Assured Clouds", *Journal of Internet Services and Applications*, volume 3, number 1, November 2011.
3. John Bellessa, Evan Kroske, Reza Farivar, Mirko Montanari, Kevin Larson, and Roy H. Campbell, "NetODESSA: Dynamic Policy Enforcement in Cloud Networks", *30th IEEE Symposium on Reliable Distributed Systems Workshops (SRDS 2011)*, Madrid, Spain, October 4-7, 2011.
4. Cristina L. Abad, Yi Lu; and Roy H. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling", *2011 IEEE International Conference on Cluster Computing (CLUSTER 2011)*, Austin, TX, September 26-30, 2011.
5. Antonia Pecchia, Aashish Sharma, Zbigniew Kalbarczyk, and Domenico Cotroneo, Ravishankar K. Iyer, "Identifying Compromised Users in Shared Computing Infrastructures: A Data-Driven Bayesian Network Approach", *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011)*, Madrid, Spain, October 4-7, 2011.
6. Mirk Montanari and Roy H. Campbell, "Attack-resilient Compliance Monitoring for Large Distributed Infrastructure Systems", *5th International Network and System Security (NSS 2011)*, Milan, Italy, September 6-8, 2011.
7. Brian Cho and Indranil Gupta, "Budget-Constrained Bulk Data Transfer via Internet and Shipping Networks", *8th International Conference on Autonomic Computing (ICAC 2011)*, Karlsruhe, Germany, June 14-18, 2011.

2010

1. Brian Cho and Indranil Gupta, "New Algorithms for Planning Bulk Transfer via Internet and Shipping Networks", *IEEE 30th International Conference on Distributed Computing Systems (ICDCS 2010)*, Genoa, Italy, June 21-25, 2010.

Theses

1. Carlo Di-Giulio, "Privacy and Security in the Clouds: IT Security and Privacy Standards in the EU and US", MA Thesis, School of Information Science, University of Illinois at Urbana-Champaign, May 2017.
2. Zachary Estrada, "Dynamic Reliability and Security Monitoring: A Virtual Machine Approach", PhD Thesis, University of Illinois at Urbana-Champaign, August 2016.
3. Mohammad Ahmad, "Cauldron: A Framework To Defend Against Cache-Based Side-Channel Attacks In Clouds", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2016.
4. Reza Shiftehfar, "A Flexible Fine-grained Adaptive Framework for Parallel Mobile Hybrid Cloud Applications", PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, December 2015.
5. Uttam Thakore, "A Quantitative Methodology for Evaluating and Deploying Security Monitors," MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, August 2015.
6. Phuong Cao, "An Experiment Using Factor Graph for Early Attack Detection", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
7. Mayank Pundir, "ZORRO: Zero-cost Reactive Failure Recovery in Distributed Graph Processing", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
8. John Bellessa "Implementing MPLS with Label Switching in Software Defined Networks", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
9. Gourav Khaneja, "An Experimental Study of Monolithic Scheduler Architecture in Cloud Computing Systems", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
10. Xiao Cai, "Phurti: Application and Network-Aware Flow Scheduling for MapReduce", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
11. Gary L. Wang, "Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring", MS Thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, May 2015.
12. Weijie Liu, "Inter-Flow Consistency: Novel SDN Update Abstraction For Supporting Inter-Flow Constraints", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.
13. Fangzhou Yao, "Secure Framework for Virtualized Systems with Data Confidentiality Protection", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, December 2014.
14. Peter Dinges, "Symcretic Testing of Programs", PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, December 2014.
15. Cristina Abad, "Big Data Storage Workload Characterization, Modeling and Synthetic Generation", PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, March 2014.

16. Jon Grov, "Transactional Data Management for Multi-Site Systems", PhD Thesis, University of Oslo, January 2014.
17. Mirko Montanari, "Limiting Information Exposure in Multi-domain Monitoring Systems", PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, July 2013.
18. Parya Moinzadah, "I-AdMiN: A Framework for Deriving Adaptive Service Configuration in Sensor Networks", PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, December 2013.
19. Qiaomin Xie, "Routing and Scheduling for Cloud Service Data Centers", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, December 2012.
20. Brian Cho, "Satisfying Strong Application Requirements in Data-Intensive Cloud Computing Environments", PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, October 2012.
21. Tobias Muhlbauer, "Formal Specification and Analysis of Cloud Computing Management," MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, January 2012.
22. Jonas Eckhardt, "Security Analysis in Cloud Computing Using Rewriting Logic", MS Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, January 2012.
23. Musab Al-Turki, "Rewriting-based Formal Modeling, Analysis and Implementation of Real-time Distributed Services", PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, August 2011.

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

ACC Assured Cloud Computing

API Application Programing Interface

APTs Advanced Persistent Threats

C&C Command and Control

CAP Capture

CCap Compromised Capacity

CCM Cloud Control Matrix

CPU Central Processing Unit

CSA Cloud Security Alliance

CSP Cryptographic Service Provider

DalvikVM is a discontinued process virtual machine in Google's Android operating system that executes applications written for Android.

DBSCAN Density-based spatial clustering clustering of applications with noise

DISA Defense Information Systems Agency

DoD Department of Defense

DoS Denial of Service

FedRamp Federal Risk and Authorization Management Program

FIFO First in, first out

GOSHED Guest OS Hang Detection

HAV Hardware Assisted Virtualization

HRKD Hidden RootKit Detection

HVAC Heating, ventilation and air conditioning

IDS Intrusion Detection Systems

IMCM Illinois Mobile Cloud computing Manager

Intel's CAT Technology Cache Allocation Technology

I/O Input/Output

IP Internet Protocol

IPY Interoperability & Portability

ISO/IEC International Organization for Standardization/International Electrotechnical Commission

ISP Internet Service Provider

IT Information Technology

JSQ-MaxWeight algorithm is a heavy-traffic optimal only for a special traffic scenario with two locality levels.

KVM Kernel-based Virtual Machine

LAMP stack is a popular open source web platform commonly used to run dynamic web sites and servers.

LANL Los Alamos National Lab

LFGraph is a recent study by UIUC which seems solid and promises the best results in distributed graph analytics.

LKM Linux Loadable Kernel Module

LTL Linear Temporal Logic

MLSSystems Multilevel Security

MongoDB is a free and open-source cross-platform document-oriented database program.

MOS Mobile Security

NCSA National Center for Supercomputing Applications

NetFlow is a network protocol developed by Cisco for collecting IP traffic information and monitoring network traffic.

NIST SP National Institute of Standards and Technology Special Publication

NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

OpenSSL Secure Sockets Layer is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used in internet web servers, serving a majority of all web sites.

OS Operating System

PCA Principal Component Analysis

PCAP Packet Capture

PED Privilege Escalation Detection

PEP Policy Enforcement Point

PDP Policy Decision Point

PMM Policy Manager Machine

PVeStA A Parallel Statistical Model Checking and Quantitative Analysis Tool

Raik Database is a line of distributed databases is built on a set of core services providing a highly reliable, scalable distributed systems framework.

RAMP Cloud Storage System

SALSA Language Simple Actor Language System and Architecture programming language is an actor-**oriented** programming language that uses concurrency primitives beyond **asynchronous** message passing, including token-passing, join, and first-class continuations.

SDNs Software Defined Networks

SDX Software Defined Internet Exchange

SLAs/SLOs Service Level Agreements/Objectives

SOC2 report focuses on a business's non-financial reporting controls as they relate to security, availability, processing integrity, confidentiality, and privacy of a system.

SQL Structured Query Language

ssh Secure Shell

SSHD Solid State Hybrid Drives

TCIP Transmission Control Protocol

TPC Transmit Power Control

TSPC Trust Services Principles and Criteria

URL Uniform Resource Locator

vCPU Virtual Central Processing Unit

VMs Virtual Machines

VMI Virtual Mobile Infrastructure

XACML eXtensible Access Control Markup Language

YARN is the architectural center for Hadoop that allows multiple data processing engines such as inter as interactive SQL, real-time streaming, data science and batch processing to handle data stored in single platform, unlocking an entirely new approach to analytics.

ZooKeeper fault-tolerant distributed key/value data store